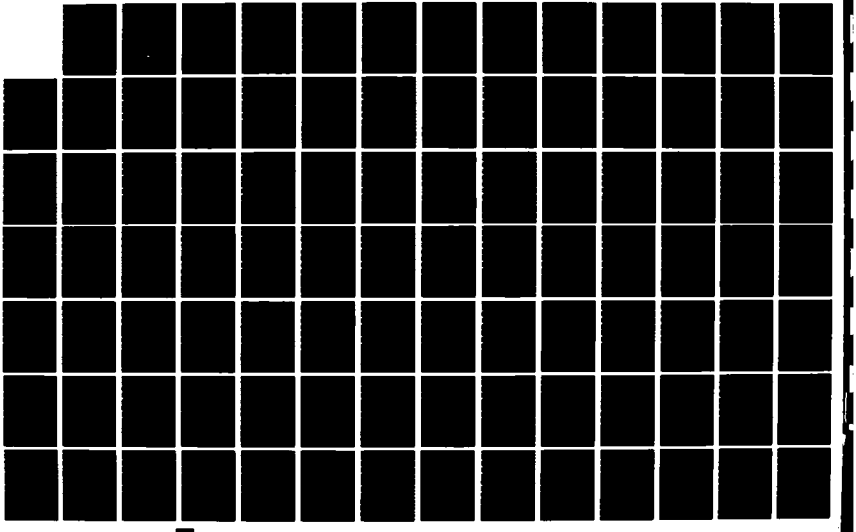
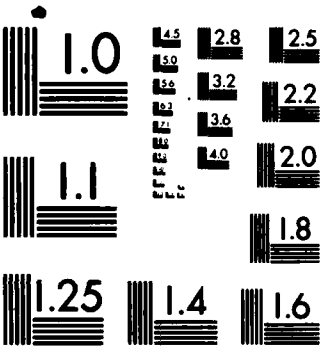
AD-A160 999 THE USE OF CK-LOG FORMALISM FOR KNOWLEDGE REPRESENTATION AND PROBLEM SOLV. (U) RUTGERS - THE STATE UNIV NEW BRUNSWICK NJ LAB FOR COMPUTER SC

1/2

UNCLASSIFIED C V SRINIVASAN 30 SEP 85 NRL-8902 F/G 9/2 NL





NRL Report 8902

The Use of CK-LOG Formalism For Knowledge Representation and Problem Solving in OPPLAN-CONSULTANT: An Expert System for Naval Operational Planning

C. V. SRINIVASAN

Navy Center for Applied Research in Artificial Intelligence
and
Department of Computer Science,
Rutgers University, New Brunswick, N.J.

September 30, 1985

NAVAL RESEARCH LABORATORY
Washington, D.C.

AD-A160 999

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NRL Report 8902	5. MONITORING ORGANIZATION REPORT NUMBER(S) NRL Report 8902
6a. NAME OF PERFORMING ORGANIZATION Laboratory for Computer Science Research 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Rutgers University New Brunswick, NJ 08930	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Office of Naval Research 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER Contract #N00014-82-C-2126
8c. ADDRESS (City, State, and ZIP Code) Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 61101E; PROJECT NO.; TASK NO. 4315; WORK UNIT ACCESSION NO. DN 280-157

11. TITLE (Include Security Classification)
The Use of CK-LOG Formalism for Knowledge Representation and Problem Solving in OPPLAN-CONSULTANT: An Expert System for Naval Operational Planning

12. PERSONAL AUTHOR(S)
Srinivasan, C.V.*

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM 1982 TO 1984	1985 September 30	113

16. SUPPLEMENTARY NOTATION
*Currently on sabbatical leave from Rutgers University

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Operational planning; CK-LOG; Knowledge representation; Problem solving

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This report introduces the principal concepts in the organization and operation of the logic-based knowledge processing system, called CK-LOG (A Calculus for Knowledge in LOGic). CK-LOG uses the *frame*-based system MDS (the Meta Description System) for knowledge representation and for modelling world states. It uses an inference engine based on *Natural Deduction* for stating and solving problems. The operation of CK-LOG is discussed here in the context of OPPLAN-CONSULTANT, an expert consultation system for Naval Operational Planning that is now being designed using the CK-LOG formalism. CK-LOG itself is currently under implementation. This report establishes the *descriptive adequacy* of CK-LOG to state and represent military planning knowledge needed for operational planning, as well as its *problem solving adequacy* to use this knowledge automatically to help military commanders design operational plans to achieve given military objectives. The organization of the planning knowledge and the planning processes in OPPLAN-CONSULTANT are directly applicable to the Battle Management Program of the Strategic Computing Initiative.

(Continues)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL C.V. Srinivasan	22b. TELEPHONE (Include Area Code) (202) 767-2381 22c. OFFICE SYMBOL 7510

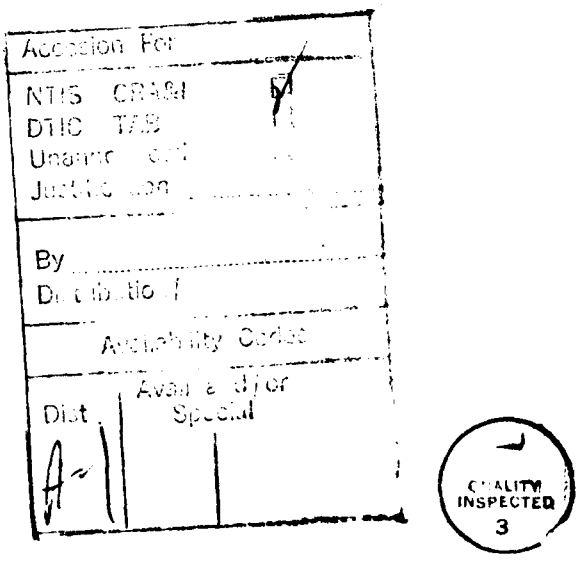
DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

19. ABSTRACT (Continued)

As a knowledge processing system, CK-LOG has several capabilities which are new to the technology of knowledge representation systems: CK-LOG has special facilities to represent and reason about actions and their time dependencies. Actions that occur in a world state may create or destroy objects in the world or modify their properties, or prevent or support other actions. The effects of actions are described in CK-LOG using *modal operators* like CREATE, DESTROY, PREVENT, SUPPORT, KEEP, etc. These operator expressions are also used to represent and reason about the possible worlds that the actions might lead to. CK-LOG uses these capabilities in a planning problem solving context. Most significantly, CK-LOG is a logic-based knowledge processing system, just as PROLOG is a logic-based programming system. CK-LOG uses a three-valued logical system with truth values T (true), ? (unknown), and F (false) to build partial models of world states, and the two-valued logical system of T and F in its theorem-proving system. The use of the three-valued logical system in its models of world states enables CK-LOG to solve problems in the context of incomplete information about world states.

The theorem-proving system of CK-LOG uses a variant of the *calculus of sequents* first proposed by Kanger (which itself is a variant of Gentzen's system). The two variations in CK-LOG are (a) the use of a new algorithm called the *mating algorithm* for testing proof terminations, and (b) the use of specialized inference rules for reasoning about *modal expressions* using the *possible world* semantics. The mating algorithm gives the theorem-proving system of CK-LOG several new capabilities: to identify information pertinent to a given problem and retrieve it from its knowledge base, to update its models of possible worlds during the problem solving process based on the findings of the theorem-proving system, to use these models of world states to test proof terminations, and to generate hypotheses during the problem solving processes that are based on unknown information. These various features of CK-LOG are described in this report. The way they are used to solve a military planning problem is illustrated by analyzing CK-LOG's solution of a planning problem taken from NWP-11.



CONTENTS

THE USE OF CK-LOG FORMALISM FOR KNOWLEDGE REPRESENTATION AND PROBLEM SOLVING IN OPPLAN-CONSULTANT: AN EXPERT SYSTEM FOR NAVAL OPERATIONAL PLANNING

1. INTRODUCTION

1.1. The Problem and the System

The following abstract taken from NWP-11* defines the Naval Operational Planning problem:

> Based on 'mission-type' directives promulgated by the Joint Chiefs of Staff, basic war plans are developed by unified and specified commanders. The naval component commanders of unified and specified commands prepare supporting operation plans and their subordinate commanders develop detailed operational plans based on their assigned tasks.
>
> The commander charged with a mission bears sole responsibility for its satisfactory accomplishment. He is responsible for the movement, support, protection, coordination, and control of his forces. He is responsible for accomplishing his mission with minimum cost and effort. Since the directive he promulgates must lead to the accomplishment of his assigned mission, he must transmit instructions and essential information so that his subordinates can perform their tasks intelligently. In effect, the assignment of a mission to a commander confronts him with a problem to solve in which he has three major functions: recognition and interpretation of his mission and its significance; *solution of his problem* through the use of his planning resources; and exercise of his judgment through command and control measures, as planned operation is executed.

It is impossible to state this problem precisely because its interpretation presupposes a vast reservoir of common sense and military knowledge, training, practice, and human judgment. I do not believe a computer system can solve this problem automatically in the near future (within the next 20 years), but it can give valuable assistance. This report presents a particular software system, called CK-LOG (Calculus for Knowledge in LOGic, pronounced C-K-LOG or CheckK-LOG), which is now being used to implement an experimental consultation system for naval operational planning. The report introduces the principal characteristics of CK-LOG by illustrating through a planning example how it represents military planning knowledge and uses it, how it 'understands' military missions and actions, and uses its problem solving abilities to assist a human planner in the planning, plan analysis, and plan execution processes.

The complexity of a mission will depend on the command level of the commander. The following are typical of the mission statements that require planning:

'Prevent establishment of base by ... at ... during ...,'
'Regain control of Bolo Island by amphibious action by Feb. ...,'
'Protect Naval and Merchant fleet in the ... Gulf during ...,'

Manuscript approved March 4, 1985.

*Naval Warfare Publication, Volume 11 (Rev. C), Department of the Navy, Office of the Chief of Naval Operations, Washington, DC 20350. This publication is available in the Naval Warfare Publications Library.

'Give escort to the ... convoy across the Mediterranean,'
'Provide combat intelligence to ... fleet,'
'Provide logistic support to ... during ...,'
'Proceed to the staging area ... by ... before ...,' etc.

The mission statements are usually brief, and the mission satisfaction usually concerns the formation and coordination of a large number of force units and weapons. Also, the geographical area of concern to a mission can be quite large. The complexity of modern naval warfare makes it imperative that a responsible commander plan his course of actions carefully and identify in advance the best means of utilizing and protecting his available forces to achieve an assigned mission. Such planning may take a team of experts anywhere from 2 weeks to 2 months' time. The planning criteria and the planning process are well documented in NWP-11, and this work is based on the guidelines given in this publication.

NWP-11 details specifications for formulating military plans and creating operational orders, and clearly identifies the various military tasks which a commander should anticipate and plan for. But it does not give the methods for solving the planning problems that it poses. These methods are left to the individual experience, training, and judgment of the commander. NWP-11 gives standardized methods for documenting plans and generating operational orders. The standard document for an operational order is called an OPORDER, and the document for recording a plan is called an OPPLAN. A commander's understanding of an OPORDER, and the OPPLAN generated by him to achieve his assigned mission, together provide the fundamental basis for the exercise of his command. The discussion in this report will provide an introduction to the military planning process described in NWP-11.

Currently, OPORDERs and OPPLANs provide the only means of recording and transmitting a commander's intentions through a command hierarchy. Understanding the mission, identifying the relevant physical objectives, deciding on appropriate courses of action in light of known enemy capabilities, planning for these actions, and executing them are, in that order, the principal tasks faced by a commander who receives an OPORDER. CK-LOG can assist a commander in several ways:

1. help the commander to evaluate the significance of his mission in the existing world state;
2. point out the planning alternatives, missing information, assumptions, and the possible own and enemy courses of action;
3. help the commander evaluate his actions against the possible actions of his enemy;
4. point out the planning tasks for a chosen course of action and reduce these tasks to their basic elements;
5. help the commander in recording his planning decisions in a systematic way, and generate OPORDERs in the standard format; and
6. help the commander in monitoring the execution of a plan, and revise it if necessary.

To do this, CK-LOG should be capable of understanding a commander's mission, the situations in the world state in which the mission is to be achieved, the nature of friendly and enemy naval military actions and their relationships to the achievement of a given mission, and to deployment and employment of own and enemy forces. It should also be capable of communicating with its users about plans, planning goals, planning decisions, and military situations. The creation of these facilities within CK-

LOG required the solution of difficult problems in artificial intelligence. The problems pertain to the following:

- Representation of actions and implementation of methods for reasoning about them in given world states, taking into account their time dependencies.

- Representation of hypothetical worlds and their relationships to ongoing actions, and using them in a (planning) problem solving context.

CK-LOG provides these facilities within the framework of a knowledge-based problem solving system. It uses a logical framework for representing knowledge of objects, actions, and concepts, and reasoning about them in given world states, taking into account their time dependencies. Situations in world states are described in CK-LOG using statements in a logical language. CK-LOG has special facilities to describe and reason about actions and their time dependencies. Actions that occur in a world state may create or destroy objects in the world, or modify their properties, or prevent or support other actions. The effects of such actions are called *events*. They are described in CK-LOG by using *operators* like CREATE, DESTROY, PREVENT, KEEP, OCCUR, SUPPORT, etc., that operate on logical expressions.* Operator expressions are interpreted by CK-LOG as referring to the truth or falsity of their arguments in *hypothetical worlds*, the worlds in which the actions that are implicitly denoted by the expressions have terminated. For a given arbitrary logical expression, CK-LOG uses its knowledge base and inferencing facilities to identify the actions that are associated with it by a given operator. Thus, the statement

[CREATE
((EXISTS region_1 (southern-shore-of France))
((during interval_1)(controlled-by region_1 Allies)))]

specifies that in some possible future world, because of the successful completion of the actions implicitly referred to by the CREATE operator, it might be true that the Allies have control of some region in the southern-shore-of France during the specified interval.

If this statement is given to CK-LOG as a planning goal, then CK-LOG will use its inference engine, its model of the current world state, and its knowledge base to identify the actions needed to select a region in southern France and take control of it, and analyze the actions (and the possible enemy actions) in order to plan for them. Thus, statements in CK-LOG have both declarative and procedural interpretations.

These facilities of CK-LOG give the means to describe military actions, military objectives, and world states, and reason about them in a planning context. This report will describe the use of this system, the way planning knowledge is represented, and the kinds of man-machine interactions that may occur during planning. The knowledge description paradigm used in CK-LOG, along with the specific objectives and outline of this report, are introduced in the next subsection.

1.2. The Knowledge Description Paradigm

We begin with a universe U which consists of physical objects, actions, and concepts. Concepts are sets of objects and/or actions. Actions modify the objects, concepts, and actions themselves. The state of this universe at time, t, is called the *world state*, denoted by U_t.

*These operators have the status of *modal operators* in the logic used by CK-LOG. They refer to modalities of truth of logical expressions in possible future (past) worlds.

Let X be a concept in U. It is assumed that an understanding of X is determined by knowledge of the five aspects of X shown below:

Structure: Components of X, i.e., the set of elements of U that together constitute X, and the way they are related to X.

Function: What X is intended for. Statements which can be made true by X in the universe U.

Behavior: Valid statements about the modifications in U that may occur while X is performing its function.

Analysis: Generating valid statements about the *function*, (or *function* and *behavior*) of X from given specifications of *structure* (or *structure* and *function*) of X.

Design: This is the converse of analysis; generating valid statements about *structure* (or *structure* and *function*) of X, from given specifications of *function* (or *behavior* and *function*) of X.

To describe these five aspects, we need a language. The language of CK-LOG is called **TML** (The Meta Language). For a concept X in U, let **K**(X) denote the *knowledge of* X; namely, the definition of the above five aspects of X in the language **TML**. Let **K**[U] be the *knowledge of* U; namely, the set of all **K**(X) for X in U. CK-LOG is the software system that is used to represent **K**[U] in a computer, and to state and solve (planning) problems in the universe U. (CK-LOG + **K**[U]) will yield the knowledge-based problem solving system for the universe U.

As we shall see later, the definitions of *structures* of concepts in U will introduce to CK-LOG the elements of a logical language, **DL** (The Domain Language). This is the language used by CK-LOG to describe situations in the world states, U_i. It is also used to define the aspects of *function, behavior, design*, and *analysis* of the concepts in U. One may view **DL** as being a sublanguage of **TML**, without operator expressions, that is specialized to the given universe, U.

Another part of the definition of the knowledge of a given universe in CK-LOG is the definition of the *operators* that are used in **TML** to describe situations in possible future worlds. These operators are defined to CK-LOG by specifying the inference rules which CK-LOG should use to analyze them. Thus, **TML** is used not only to define the knowledge **K**[U], but also to define the domain language **DL** and inference processes for the operator expressions. Hence, the name 'The Meta Language.'

Using a logical language as the domain language gives two advantages: descriptions of **K**[U] in **TML** and **DL** have well-defined logical semantics, and one can use general logical deductive techniques to state and solve problems in the universe U, without having to write specialized programs to specify how CK-LOG should use **K**[U] in its problem solving processes. The way this is done in CK-LOG is briefly described in Section 3 (see Ref. 1 for details). The way the problem solving abilities of CK-LOG are used to create military plans is illustrated in Section 5. CK-LOG does not attempt to solve difficult problems by itself, but acts as an intelligent assistant to a human problem solver. CK-LOG uses the natural deduction system of Gentzen [2] (Section 3 and Ref. 1) for stating and solving problems, and uses models to guide itself through its problem solving processes. CK-LOG uses MDS (the Meta Description System) [1, 3-5] to model world states and its own problem solving processes.

Our universe, U, is the universe of *Naval Operational Planning*. The knowledge, **K**[U], discussed in this report, is typical of the knowledge for Naval Operational Planning. The software system (CK-LOG + **K**[U]) is the planning consultant, called OPPLAN-CONSULTANT. The representations of the knowledge of this universe in a computer, statements of the operational planning problems, and the reasoning and problem solving processes that are used by the computer to analyze the representations,

create plans, and monitor its execution, will all depend crucially on the structure and semantics of the language **DL**. The language structure is close to human intuition, yet it is not as easy to use as English. As with all logical languages, practice and familiarity are essential to converse in this language with a computer.

The statements in **TML** that appear in this report, their associated commentaries, and the planning example discussed here are together intended to achieve two purposes simultaneously: first, to demonstrate the *descriptive adequacy* of the languages **TML** and **DL**, to define the planning domain knowledge, to state the military situations in which the planning takes place, and to specify the problems that need to be solved in planning and to plan execution processes. One may argue for descriptive adequacy of **TML** based on the following observations: **TML** (and **DL**) can describe not only situations that are true in the existing worlds, but also situations that might be true in hypothetical future worlds. Also, one may describe the time dependencies among these worlds, as well as the manner in which these hypothetical future worlds are related to the current worlds through ongoing actions. These provide the foundation for describing military actions, their objectives, and their effects on world states.

The second purpose is to demonstrate the *problem solving adequacy* of the system to interpret statements in these languages, to make use of specified domain knowledge, to set up planning goals, to propose alternatives for actions, to identify missing information needed for planning, and to evaluate proposed plans. A convincing demonstration of the adequacy of these methods for the development of planning consultants for operational planning should await complete implementation of OPPLAN-CONSULTANT and its testing in realistic environments. The current implementation is intended to be an initial demonstration of the feasibility of using CK-LOG for this purpose.

This report does not cover all details of the operational planning process and the OPPLAN-CONSULTANT that is needed to support it. Much of this still remains to be identified and defined. The methods presented here are illustrative of the kinds of knowledge and the kinds of problem solving processes that one can specify in CK-LOG. The knowledge description paradigm presented above is used to describe the concepts of military action (which I shall call MIL-ACTION), OPORDER, OPPLAN, COMMANDER, FORCE, and other allied concepts in the universe of naval operational planning as illustrated in Section 2. The way **TML** is used in Section 2 to describe these concepts is precisely the way one would specify them to CK-LOG, except for some minor syntactic differences The definitions of the concepts in **U** presented here are typical of the complex domain knowledge structures which one may specify to CK-LOG with relative ease, in a purely declarative format.

The details given in Section 2 capture enough of the military planning knowledge specified in NWP-11 for later introduction of a planning problem and its solution. This should also provide a reasonably convincing demonstration of the descriptive adequacy of **TML** and **DL**. The planning problem is taken from NWP-11, Appendix F, introduced in Section 4, and its solution discussed in Section 5. The material in Section 5 illustrates the general methods used by CK-LOG to pose planning problems and solve them. It gives a partial demonstration of the problem solving adequacy of CK-LOG for operational planning.

The nature of the CK-LOG system and its inferencing processes are introduced in Section 3. Here is a highly simplified abstract of discussions that follow in the ensuing sections:

A military action has military objectives and plans. Its function is to achieve the military objective by executing the action, and its behavior is that it might destroy the forces that are engaged in the action and might result in the forces controlling different geographical regions. The control of regions may transfer from one force to another as a result of the military action. The commander is responsible for achieving the objective of the military action. He exercises this responsibility by planning for the action and by controlling the execution of the planned action. The commander is also responsible

for protecting his forces, approving plans, and issuing operational orders. These orders specify the military objectives (missions), military actions, and assign forces and commanders to the military actions. The function of an OPORDER is to cause the commander who receives it to plan for the achievement of his assigned mission and have the plan approved. The function of an OPPLAN is to cause the commander who created it to start the military actions planned in the OPPLAN, *if* and when the OPPLAN is declared to be active.

Having described these ideas to CK-LOG, and the structures of FORCE and REGION (geographical regions), the planning example is introduced in Section 4 by stating a military situation and the assigned missions (and their purposes). Section 5 shows how CK-LOG automatically assimilates the stated military situation into its world state model, and thereafter, starting from the simple goal statement '(ACHIEVE (objective-of opplan))' where opplan is the plan to be created, CK-LOG automatically analyzes the given objective-of opplan and generates planning alternatives that lead to the commander's decision on actions to be taken.

The report concludes with a summary of the significant contributions made by CK-LOG to knowledge representation and knowledge processing, and its potential for being used to develop complex *expert systems* like OPPLAN-CONSULTANT, with capabilities for *deep reasoning* as opposed to *evidential reasoning* as in MYCIN [6] and PROSPECTOR-[7] like systems.

2. THE DESCRIPTIVE ADEQUACY: THE UNIVERSE, U, FOR OPERATIONAL PLANNING

2.1. The Concept of Military Action

Let us begin with the concept of MIL-ACTION. The name MIL-ACTION will denote to the system the set of all possible military actions. In this sense it is a concept. At any instant of time, t, the world state, U_t may contain zero or more distinct instances of MIL-ACTION occurring in it. The *structure* of MIL-ACTION, shown below in Fig. 2.1, is a set of relational forms together with certain conditions which they should satisfy.

The statements that appear in Fig. 2.1 are typical of the statements in the meta-language, TML. Each structure definition in Fig. 2.1 introduces one or more component concepts of MIL-ACTION, and possibly also a *relation name* that is used to indicate the relationship between the parent concept and the component(s). It is important to note that the relation names are not *a priori* built into CK-LOG. They are introduced to CK-LOG through these definitions. I have here chosen names that attempt to faithfully reflect the role played by the component concept in the parent concept.

The meaning of each of these relations is defined to CK-LOG in two ways: the first is by the specification of the conditions (discussed in Section 2.1.2) which the relations should satisfy in a world state. The second is by the specification of the actions needed to make these relations true in a world state (the CREATE action for the relation), to maintain its truth (the KEEP action for the relation), or to destroy its truth (the DESTROY action for the relation). The forms of these action definitions are discussed in Section 5.

The significance of the definitions in Fig. 2.1, the linguistic forms in DL that they imply, and the interpretation given to these forms by CK-LOG are discussed in this subsection. It is assumed that the operators that are used in DL for the OPPLAN-CONSULTANT domain have been already defined to CK-LOG. Their definitions are discussed in Appendix B.

2.1.1. The Structure of MIL-ACTION

The first structure definition in Fig. 2.1 is the tuple,

(MIL-ACTION FORCE FORCE REGION).

Structure: (MIL-ACTION FORCE FORCE REGION)
(prosecuted-by MIL-ACTION FORCE), 1
(is-against MIL-ACTION FORCE), 1
(region-of MIL-ACTION REGION), 1
(ordered-by MIL-ACTION OPORDER), 1
(specified-by MIL-ACTION OPORDER-TASK), 1
(objective-of MIL-ACTION MIL-OBJECTIVE), 1
(purpose-of MIL-ACTION MIL-OBJECTIVE), 1
(mil-plan-for MIL-ACTION OPPLAN), 1
(commander-of MIL-ACTION COMMANDER), 1
(phy-objectives-of MIL-ACTION OBJECTS), 1
(supports MIL-ACTION MIL-ACTIONS), *Irreflexive*
(opposes MIL-ACTION MIL-ACTIONS), *Irreflexive*,
(execution-of MIL-ACTION MIL-EXECUTION), 1
(status-of MIL-ACTION MIL-STATUS), 1
(risk-factor-of MIL-ACTION RISK-FACTOR), 1

Function: [(*time-of* (ACHIEVE (execution-of mil-action)))
([ISTRUE (objective-of mil-action)] AND
[NOT (OCCURS (PREVENT (purpose-of mil-action)))])]

Behavior: [(EXISTS int INTERVAL)
[((during int)(MIL-ACTION x y region)) IMPLIES
([(EXISTS force FORCE)
((between int)
([OCCURS (DESTROY force)] AND
[(belongs-to force x) OR (belongs-to force y)]))]
AND
[(EXISTS r1,r2 (is-within region))
((between int) ((controls x r1) AND (controls y r2)))]))
AND
[OCCURS (ACHIEVE (execution-of mil-action)))])]]

Analysis: [ACHIEVE (execution-of mil-action)]

Design: [ACHIEVE (*design* (mil-plan-for mil-action))]

Fig. 2.1—The concept of military action, MIL-ACTION

This declares that a MIL-ACTION always involves two FORCEs and a REGION, where FORCE and REGION are also concepts in the universe U. These are the components that give a MIL-ACTION its unique identity: two military actions at a given time are different, if and only if, at least one of these components is different for them. Having made this declaration, one may now use in CK-LOG expressions of the form,

(**t-exp** (MIL-ACTION $force_1$ $force_2$ region)),

to denote the occurrence of a military action between the particular FORCEs, $force_1$ and $force_2$ in the REGION, region, at the time instant (or time interval) specified by the time-expression, **t-exp**. The entire expression shown above is a *proposition* in the language **DL**; i.e., it will have a truth value, T (*true*), F (*false*), or ? (*unknown*), in the world states, U_t. If the truth value is T, then it would mean that a military action between the said forces is occurring at the time in the said region; if it is F, then it is not occurring; and if it is ?, then it is not known whether or not it is occurring. If no time parameter is given, then the expression refers to its truth value in the *current world state*. Propositions like these, with or without time parameters, are called *atoms*. They are the basic constructs (atoms) of the language **DL**. An atom with or without the negation operator, NOT, is called a *literal*.

A **t-exp** is either a time instant, t, or an interval, $[t_1\ t_2]$, or an expression of the form (at **t-exp**), (after **t-exp**), (before **t-exp**), (during **t-exp**), (between **t-exp**), (*time-of event*), or (*interval-of event*).

The remaining parts of the structure definition enumerate the components of a MIL-ACTION and show how they are related. Thus, for example, the structure

(prosecuted-by MIL-ACTION FORCE)

declares that FORCE is a component of MIL-ACTION, and it is related to MIL-ACTION *via* the relation name, 'prosecuted-by.' In CK-LOG, this specification is interpreted as saying that a MIL-ACTION is prosecuted-by *at most* one FORCE. We need to specify, however, that there should be a unique FORCE. This is done by the declaration,

(prosecuted-by MIL-ACTION FORCE), 1

shown in Fig. 2.1, where a '1' appears after the comma. The '1' here is interpreted as specifying that there should be *at least* one FORCE. These two restrictions together imply that there should be *exactly* one such force.

Having made this declaration, one may use the phrase, '(prosecuted-by MIL-ACTION),' in the language **DL** to denote the concept, FORCE:

(prosecuted-by MIL-ACTION) = {FORCE}.

For a particular MIL-ACTION, say, mil-action and a particular FORCE say, force, the expression,

(**t-exp** (prosecuted-by mil-action force)),

is an atomic proposition in the language **DL**. Its truth value in a world state U_t will be T (if it is *true*), F (if it is *false*), or ? (if it is *unknown*). Also,

(prosecuted-by mil-action) = {force},

where the force is such that the truth value of '(prosecuted-by mil-action force)' is T or ? in the *current* world state. If the truth value is ?, then there can be more than one such force in the current world state for which the ? truth value applies. Then '(prosecuted-by mil-action)' will denote the set of all such

FORCEs. This set may be interpreted as the set of possible candidate FORCEs, one of which may be the force that prosecutes the mil-action. If the truth value is F for all FORCEs, then (prosecuted-by mil-action) will be equal to NIL (the *empty set*).

Note that throughout this report, upper case words like MIL-ACTION, FORCE, etc. denote concepts, lower case words for names of relations, and lower case small letter words like mil-action, force, etc., with or without subscripts, denote particular (or generic) instances of concepts.

The remaining structure declarations in Fig. 2.1 similarly define the phrases shown in Fig. 2.2, where OPORDER stands for the concept of an Operational Order, OPPLAN is an Operational Plan, and OPORDER-TASK is a task specified by the OPORDER, etc. The OPORDER-TASK itself will be specified by the OPORDER that orders the MIL-ACTION. The MIL-OBJECTIVEs that specify the objective and purpose of a military action will be given by this oporder-task, as also the forces that prosecute the military action. The MIL-STATUS will specify the status of the military action: whether it is ongoing or not; if it is, then the losses and casualities encountered by own and enemy forces in the action, the region controlled by the forces, the well-defended regions, regions under attack, etc., and if it is not, then its readiness, etc. The RISK-FACTOR will specify the risks involved in the action and the chances for destroying the enemy forces. These may be high, not-high, not-low, low, etc. They will depend on whether the region of the action is well-defended by the enemy or not. The structures used to represent these data in CK-LOG are not presented in this report.

(is-against MIL-ACTION) = {FORCE},
(region-of MIL-ACTION) = {REGION},
(ordered-by MIL-ACTION) = {OPORDER},
(mil-plan-for MIL-ACTION) = {OPPLAN},
(commander-of MIL-ACTION) = {COMMANDER},
(specified-by MIL-ACTION) = {OPORDER-TASK},
(objective-of MIL-ACTION) = {MIL-OBJECTIVE},
(purpose-of MIL-ACTION) = {MIL-OBJECTIVE},
(status-of MIL-ACTION) = {MIL-STATUS},
(risk-factor-of MIL-ACTION) = {RISK-FACTOR}, etc.

Fig. 2.2—Phrases defined by the structure of MIL-ACTION

Let me explain the distinction between the objective and purpose as it is enunciated in NWP-11: except in certain cases where the execution of a military action unambiguously follows from the objective statement, the purpose and objective of a military action will be different from each other. The purpose is usually the objective of a superior commander's military action. For example,* the objective of a subordinate commander might be:

'Seize and occupy the X-ray area on the southern shore of the continent of Europe.'

And his purpose, which is his superior commander's objective, might be:

'Support allied operations on the continent of Europe.'

Each commander receiving an OPORDER, x, will have a specific assignment to one of the OPORDER-TASKs specified in x. The commander will start a planning activity to achieve the MIL-OBJECTIVE, y, of his assigned task. The result of this planning activity will be a new OPORDER, z, such

* All the examples cited in this report are taken from NWP-11.

that the tasks specified by z can together achieve the objective y. The commander will now issue z to his subordinates to whom the tasks of z have been assigned. This will in turn cause the subordinates to begin their own planning activities to achieve their respective assigned objectives. This process will iterate through successive levels of a command hierarchy and terminate at the lower levels of the hierarchy, when the orders become sufficiently unambiguous that they could be executed directly without need for planning. I will capture this structure of OPORDERs and planning activities, and their intended meanings, as I proceed further with the definitions of the various concepts in the universe U.

There are various kinds of MIL-OBJECTIVEs, such as DESTROY-OBJTVE, SUPPORT-OBJTVE, BLOCKADE-OBJTVE, CAPTURE-OBJTVE, TRANSPORT-OBJTVE, etc. Corresponding to these different objectives, there are also have different kinds of MIL-ACTIONS. These different kinds of objectives and actions are called *specializations*. Thus, a DESTROY-OBJTVE will be a specialization of a MIL-OBJECTIVE, and so also a DESTROY-ACTION will be a specialization of MIL-ACTION. In CK-LOG, a specialization, X, of a concept, Y, will inherit from Y all of its *structure*, and share with Y some or all of its *function, behavior, analysis*, and *design* aspects. In addition, X may also have its own specialized additional *structures, function*, etc. This is illustrated in Fig. 2.3. Examples of specializations occur in the description of FORCE shown in Appendix A and in the description of OPPLAN in Section 2.3.

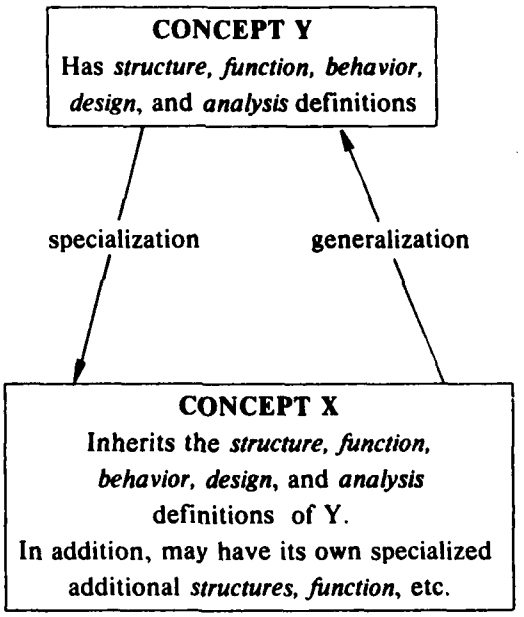


Fig. 2.3—Concept of specialization and generalization

Whereas objectives are concepts of actions, the physical objectives of a MIL-ACTION are OBJECTs (stands for physical objects). The phrase (phy-objectives-of mil-action) will denote the set of physical objects that are associated with a military action. For example, '*Neutralize enemy air forces on Bolo Island*' might be the objective, for which its associated physical objective would be Bolo Island itself, or the enemy air base on the island (see NWP-11 for more discussion on these concepts). Note that the relationship between objective and physical objectives will not be always as direct as is the case in this example. The determination of the phy-objectives-of a MIL-ACTION from the specification of an OPORDER is a part of the planning problem. These are further discussed in Sections 4 and 5.

Notice that I have used the plural form, OBJECTs, in Fig. 2.1 to indicate that there may be more than one OBJECT, x, y, z, etc., such that

(phy-objectives-of mil-action x),
(phy-objectives-of mil-action y),
(phy-objectives-of mil-action z), etc.

are all true in a world state for a military action mil-action. The declaration,

(phy-objectives-of MIL-ACTION OBJECTs), 1

in Fig. 2.1 specifies that there should be *at least* one physical objective for each military action.

A MIL-ACTION may support more than one MIL-ACTION, and the phrase (supports mil-action) will refer to the set of all MIL-ACTIONs supported by a MIL-ACTION. The restriction that a MIL-ACTION can support only other MIL-ACTIONs that are different from itself is declared to CK-LOG by the statement,

(supports MIL-ACTION MIL-ACTIONs), *Irreflexive*,

where '*irreflexive*' specifies that the relation is irreflexive, i.e., (supports x x) is always false for all x. Similarly, we also have the 'opposes' concept: a MIL-ACTION may oppose other MIL-ACTIONs. The actions opposed are referred to by the phrase (opposes mil-action). It is also irreflexive. The (execution-of MIL-ACTION) is the concept called MIL-EXECUTION, which stands for military action execution. Every MIL-ACTION will have exactly one MIL-EXECUTION associated with it. It will specify the rules for deployment and engagement of the military action. This concept is not further defined in this report.

The structure of MIL-ACTION shown in Fig. 2.1 does not specify all the conditions that the structure should satisfy in order to be consistent with the structures of its component concepts. A particular mil-action may have its occurrence described by (MIL-ACTION x y r), where x is the force that prosecutes the action, y is opposing force, and r is its region of occurrence. For any such mil-action, it should be true that the oporder-task associated with it should be one of the tasks specified by the oporder that orders the mil-action. The objective and purpose of this oporder-task should be the same as those specified in the mil-action. Formal statements of these conditions in the language DL are shown in the next subsection.

2.1.2. Restrictions on the Structure of MIL-ACTION

Each restriction presented below pertains to a relational form shown in Fig. 2.1. For each restriction, an English statement of the restriction is given followed by its logical statement in the language DL. This is then followed by a brief discussion of the structure of the statements in DL, where relevant.

Notice that, in the statements shown below, names in lower case small letters are used to denote generic instances of concepts. Thus, mil-action refers to a generic instance of MIL-ACTION. In the logical statements it has the status of a *variable*. The statements use the phrase, (specified-by mil-action), to denote the oporder-task associated with the mil-action: every mil-action has a unique oporder-task associated with it. Similarly, they use (specifies oporder-task) to denote the mil-action associated with the oporder-task: Each oporder-task also has a unique mil-action associated with it. Clearly, if (specified-by mil-action) is oporder-task, then (specifies oporder-task) is mil-action. Thus, in any world state, U_t, it is true that (read 'IFF' below as 'if and only if')

[(specified-by mil-action oporder-task) IFF (specifies oporder-task mil-action)].

The relations "specified-by" and "specifies" are said to be *converses* of each other. This is further discussed in Section 1.1.2 of Appendix A.

MIL-RESN1: For any mil-action, (MIL-ACTION x y r) iff x is the force that prosecutes the mil-action, the action is against y, it occurs in the region r, and y is an enemy of x. The logical statement of this condition is shown below:

[(MIL-ACTION x y r) IFF
([EXISTS mil-action MIL-ACTION]
[prosecuted-by mil-action x] AND
[is-against mil-action y] AND
[region-of mil-action r] AND [enemy-of x y])]

Notice that this logical expression has no time parameter. This means that it should be satisfied in every world state. The same holds true for all the restrictions shown below where the logical conditions do not contain any time parameters. Here 'enemy-of' is a relation name that applies to FORCEs: a force can be the enemy of another force (see definition of FORCE in Fig. A6 in Appendix A).

MIL-RESN2: The oporder-task of a mil-action should be a task which is specified by the oporder that orders the mil-action:

(specified-by mil-action (specifies-tasks (ordered-by mil-action)))

Here, the oporder-task that specifies the mil-action is denoted by (specified-by mil-action). This oporder-task is being declared to be the same as (specifies-tasks (ordered-by mil-action)). Here, '(ordered-by mil-action)' refers to the oporder of the mil-action. An oporder may specify several tasks. Thus, '(specifies-tasks oporder)' is a set of OPORDER-TASKs. The relational statement above is interpreted as saying that *there exists an* oporder-task *in* (specifies-tasks (ordered-by mil-action)) *such that* (specified-by mil-action oporder-task) *is true*. This may be written explicitly in the language DL as follows:

[(EXISTS oporder-task (specifies-tasks(ordered-by mil-action)))
(specified-by mil-action oporder-task)]

MIL-RESN3: In the following, the phrase '(specified-by mil-action)' denotes the oporder-task of the mil-action. This restriction says that the objective of a mil-action is the same as the objective of its oporder-task:

(objective-of mil-action (objective-of (specified-by mil-action)))

MIL-RESN4: The purpose-of mil-action is the same as the purpose-of its oporder-task:

(purpose-of mil-action (purpose-of (specified-by mil-action)))

MIL-RESN5: A mil-action is prosecuted-by the forces assigned to it by its oporder-task:

(prosecuted-by mil-action (assigned-forces (specified-by mil-action))).

MIL-RESN6: If the oporder-task of mil-action supports another OPORDER-TASK, y, then mil-action supports the military action that y specifies:

[(EXISTS y OPORDER-TASK)
((supports (specified-by mil-action) y)) IMPLIES
(supports mil-action (specifies y)))]

The details on how restrictions like these are specified to CK-LOG are not given here (see Ref. 1 for details). They are stated using logical statements like the ones shown above.

2.1.3. The Function, Behavior, Analysis, and Design of MIL-ACTION

The *function* in Fig. 2.1 uses the *operators*, ACHIEVE, ISTRUE, and PREVENT. Let me briefly explain what these operators refer to and what they mean, i.e., when they become true (they are discussed again in Section 2.4 and Appendix B). (ACHIEVE (execution-of mil-action)) refers to the set of processes that may be invoked to perform the said executions. It will be true in a world state when all the processes involved in the execution terminate successfully in that world state. (ISTRUE (objective-of mil-action)) declares that the objective should be true in the world state at the time this is evaluated. (PREVENT (purpose-of mil-action)) refers to the processes that might prevent the achievement of the purpose-of the mil-action, and (NOT (OCCURS (PREVENT ...))) says that these processes should not occur. The expression '(*time-of* (ACHIEVE (execution-of mil-action)))' refers to the time instant when the (ACHIEVE (execution-of ...)) becomes true.

The *function* in Fig. 2.1 may now be paraphrased as follows: at the time the execution-of the mil-action is achieved (i.e., completed successfully), its objective should be true in the world state and its purpose should not be prevented. Notice that this *function* definition does not specify when the execution itself should be performed. This is specified later in the OPPLAN definition (Section 2.5): the execution of a mil-action will commence only when the status-of its OPPLAN becomes active. CK-LOG will use the *function* statements like this to set up goals for itself in its problem solving activities. This is further discussed in Sections 2.6 and 5.

Note that an expression of the form, '[(*time-of* x) y]' (or '[(after (*time-of* x)) y]') where x and y are logical expressions, establishes a *causal connection* between the statements x and y: making x true should cause y to become true at the same time (or after that time). In any such causal connection, the logical assertion '[x IMPLIES y]' will be true. But, the causal assertion makes a stronger statement than the implicational assertion, because it also specifies a time relationship.

The *behavior* of a MIL-ACTION shown in Fig. 2.1 makes a general statement about what happens in a mil-action. Here int is a time INTERVAL that is specified by a pair of TIME points, (t_1, t_2). The statement says the following: if there is an int during which (MIL-ACTION x y region) is true, for any x, y, and region, then there are DESTROY processes against FORCEs belonging to x and y that will OCCUR between the time points of the same int, and there will exist REGIONs within region that are controlled by the participating forces x and y between the same time points. The expression, '(during int)' is interpreted as 'for all time instants, t, in the semiclosed interval, $[t_1, t_2)$,' and '(between int)' is interpreted as 'there exists time instants, t, in the open interval, (t_1, t_2).'

The DESTROY operator may operate on instances of concepts, like force above, or on logical expressions containing particular or generic instances of concepts. When it operates on an instance, it refers to the processes, P, that might be needed to destroy, i.e., make false, *all* or an *a priori specified subset* of the properties of the instance. When it operates on a logical expression, it refers to the processes, P, that might be needed to make the expression false. It will be true (become true) in a world state if the specified properties are (become) false in that world state. If the specified properties

are not false and P is not NIL, then the truth value of (DESTROY ...) will be ? and that of (OCCURS (DESTROY ...)) will be T, because the processes, P, are occurring. If P is NIL, then (OCCURS (DESTROY ...)) is F. If the specified properties are not false and P is NIL, then the truth value of (DESTROY ...) is F. Notice that (DESTROY ...) may be true in a world state (because the specified properties are false) while (OCCURS(DESTROY ...)) is false (because P is NIL). Similar considerations apply also to CREATE, ACHIEVE, and other operators used in this report; these will attempt to make the specified properties true. I will say more on the OCCURS operator, and the distinction between CREATE and ACHIEVE in Section 2.4.

Note that the behavior statement in Fig. 2.1 does not say that there should necessarily be destruction of FORCEs: (OCCURS (DESTROY force)) simply says that DESTROY processes against force occur. The DESTROY processes may not succeed: if the processes started by DESTROY all terminate without making the specified properties false, then destruction would not have occurred. The use of 'between' instead of 'during' indicates that the DESTROY processes need not occur all through the int. Similarly, the use of 'between,' in the statement about regions controlled, allows the regions controlled to change at different times in the int. Thus, the behavior statement is a general statement that captures the nature of a MIL-ACTION: in all world states, if a military action occurs, then this is what one may expect.

In a planning environment, *behavior* statements like these will be used by CK-LOG to anticipate the contingencies which should be accounted for in its plans. The above *behavior* statement may thus be used by the system to set up planning subgoals to protect a commander's forces as a part of a military planning process, since it knows that every commander has to keep his forces operational (see Section A1 in Appendix A), and to keep them operational, their destruction, should be prevented. A standard way of destroying an action is to make its *behavior* false. Thus, to prevent the destruction of the forces of a commander, CK-LOG will set up goals for the commander to make the *behavior* of actions that seek to destroy the forces, and goals for preventing occurrences of situations in which such actions may occur. We see applications of this in Section 5.

The *analysis* definition says that to analyze a mil-action, its execution should be analyzed; and the *design* definition says that to design a military action, the plan for the action should be designed.

To get an idea of the way this concept of MIL-ACTION is interpreted, progressively developed, and used for forming plans and monitoring execution, it is necessary to proceed further with the definitions of its component concepts. A summary of the principal relationships between MIL-ACTION, OPRODER, OPPLAN, FORCE, and COMMANDER is given in Fig. 2.4. A COMMANDER creates an OPPLAN. This give-rise-to an OPORDER which is received by a COMMANDER who is a subordinate-of the one who issues it. The OPORDER specifies several OPORDER-TASKs. One of these is shown in Fig. 2.4. This OPORDER-TASK specifies a MIL-ACTION and assigns FORCE and a COMMANDER to it. This COMMANDER is the same one that received the OPORDER. He has-command-of the assigned forces and is also the commander of the MIL-ACTION.

The objective of the OPORDER is the same as the purpose-of the MIL-ACTION, and the objective of the OPORDER-TASK is the same as the objective of the MIL-ACTION. If an OPORDER specifies, for example, five OPORDER-TASKs, then the objectives of these five OPORDER-TASKs would together be expected to achieve the objective of the OPORDER itself. The COMMANDER who is assigned by the OPORDER-TASK to command the MIL-ACTION, now creates a mil-plan-for this action. This is a new OPPLAN created by this COMMANDER, which in turn will give-rise-to a new OPORDER. Thus, the recursive process of plan generation will continue. More detailed definitions of these concepts are given in Appendix A. The function of OPORDER, the definitions of OPPLAN, and the action BUILD-BASE are discussed below.

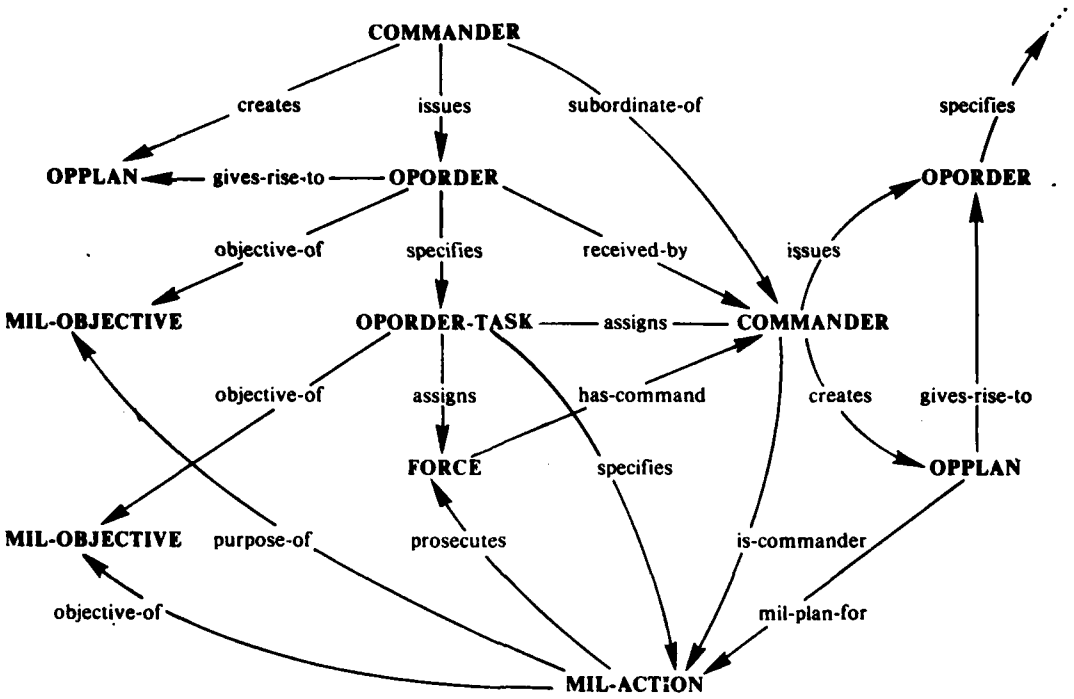


Fig. 2.4—Principal relationships between OPORDER, OPPLAN, COMMANDER, FORCE, MIL-ACTION, and MIL-OBJECTIVEs

2.2. The Function and Behavior of OPORDER

The definition of the OPORDER concept is shown in Appendix A. Let me here comment on its *function* (see Appendix A for its formal definition). The *function* of an oporder is to cause each commander of its oporder-tasks to CREATE task-plan-for his task and have it approved-by his superior, after receiving the OPORDER. The statement shown in Fig. A1 in Appendix A may be paraphrased as follows: for every oporder-task specified by eporder, the comndr, who is the assigned-comndr-of the oporder-task, should CREATE the task-plan-for the oporder-task and have it approved, after he has received the oporder. Notice that this function statement requires that the CREATE process be successful (i.e., true at some time after the receipt of the oporder by the comndr). If this CREATE process is not successful, then the oporder would not have succeeded in its function. Notice again the causal connection between receipt and creation.

The *behavior* that an oporder evokes on a world state is simply that it causes a planning process to be initiated by the commanders to whom it has been issued-to, after they have received it. The only difference between the *function* and the *behavior* statement is in the use of the OCCURS operator. The behavior is that the plan creation processes would occur. For the behavior to be satisfied, the plan creation processes need not necessarily succeed.

This kind of OCCURS relationship between *function* and *behavior* will hold for all concepts where the occurrence of an instance of the concept invariably causes the execution of its function. As we shall see in the next subsection, the same kind of relationship between *function* and *behavior* holds also for OPPLANs.

2.3. The Concept of OPPLAN, an Operational Plan

It is convenient to view an OPPLAN as a specialization of an OPORDER, for it gives essentially the same information as an OPORDER but in greater detail. Thus, an OPPLAN will inherit the structure of an

Structure: (*is-a-specialization* OPPLAN OPORDER)
(oporder-for OPPLAN OPORDER), 1, opplans-for
(oporder-task-for OPPLAN OPORDER-TASK), 1, task-plan-for
(created-by OPPLAN COMMANDER), 1
(is-mil-plan OPPLAN MIL-ACTION), 1, mil-plan-for
(planning-staff-of OPPLAN STAFF-OFFICERS)
(submitted-to OPPLAN COMMANDER), 1, has-recd-for-approval
(status-of OPPLAN OPP-STATUSs), 1, is-status
(classification-of OPPLAN SECURITY-CLN), 1
(preface-for OPPLAN OPPREFACE), 1, subject-of
(specifies-tasks OPPLAN OPPLAN-TASKs), 1, specified-by
(approved-by OPPLAN COMMANDER), 1, has-approved
(gives-rise-to OPPLAN OPORDER), 1, produced-from
(opplans-for OPPLAN) = NIL
(issued-to OPPLAN) = NIL
(issued-by OPPLAN) = NIL
(is-produced-from OPPLAN) = NIL
(orders OPPLAN) = NIL

Function: ([(after (*time-of* (status-of opplan completed)))
(CREATE
(submitted-to opplan (reports-to (created-by opplan))))]
AND
[(after (*time-of* (status-of opplan approved)))
((EXISTS x (assigned-comndr-of (specifies-tasks opplan)))
([NOT (created-by opplan x)] IFF
[(EXISTS oporder OPORDER)
([CREATE (gives-rise-to opplan oporder))] AND
[(EVERY comndr
(assigned-comndr-of (specifies-tasks opplan)))
((after (*time-of* (gives-rise-to opplan oporder)))
(CREATE (issued-to oporder comndr)))])]))]
AND
[(after (*time-of* (status-of opplan active)))
((EVERY mil-action (specifies (specifies-tasks opplan)))
(ACHIEVE ([execution-of mil-action] AND
[*function* mil-action])))])

Behavior: [OCCURS (*function* opplan)

Fig. 2.5—The OPPLAN concept

OPORDER. In addition, it will have some relational structures of its own. There are, however, certain exceptions to the inheritance rule. Not all the relational structures of an OPORDER are inherited by an OPPLAN. The 'opplans-for,' 'is-produced-from,' 'issued-by' and 'issued-to' relations do not apply to an OPPLAN. The inheritance of these relational structures is blocked by the statements,

(issued-to OPPLAN) = NIL, (is-produced-from OPPLAN) = NIL
(opplans-for OPPLAN) = NIL, (issued-by OPPLAN) = NIL, etc.

in Fig. 2.5, where NIL represents the *empty set.* The structure of an OPPLAN shown in Fig. 2.5, together with those inherited from OPORDER, captures the information given in NWP-11, Appendix A. Let me briefly review the relational structures shown in Fig. 2.5.

Every OPPLAN, opplan, is uniquely associated with an oporder and an oporder-task, where

oporder = (oporder-for opplan) = (specified-by oporder-task),
oporder-task = (oporder-task-for opplan), and
opplan = (task-plan-for oporder-task) = (opplan-of oporder).

Here, (opplan-of, oporder-for) and (task-plan-for, oporder-task-for) are both pairs of converses. Each OPPLAN is created as the task-plan-for an oporder-task. It is created by the commander to whom the oporder-task has been assigned. The OPPLAN will have the same objective and purpose as those of its oporder-task.

The status of an OPPLAN is an OPP-STATUS, which can be one of approved, pending, in-preparation, etc. It is approved if the commander, to whom it was submitted, approves it; it is pending if the approval has not been given yet, etc. The level of security classification-of an OPPLAN should be equal to or higher than the classification level of its associated *OPORDER.* The preface-for an OPPLAN is a letter that announces the existence of the plan. OPPREFACE specifies the information to be contained in this letter. This letter may have a security classification level which is less than or equal to the classification level of the OPPLAN itself, but not less than the classification level of the oporder-for the OPPLAN. Each OPPLAN may have a planning staff associated with it (planning-staff-of OPPLAN). Finally, an OPPLAN itself may give rise to an OPORDER based on the plan. This is the OPORDER that is issued by the creator of the plan to his subordinates. These subordinates will be the commanders of the military actions of the OPPLAN-TASKs of the OPPLAN.

OPPREFACE: (subject-of OPPREFACE OPPLAN), 1, preface-for
(distn-list-of OPPREFACE COMMANDERs), 1, isin-distn-list
(coordinated-with OPPREFACE COMMANDERs), participated-in
(classification-of OPPREFACE SECURITY-CLN), 1

OPPLAN-TASK: (*is-a-specialization* OPPLAN-TASK OPORDER-TASK)
(condition-of OPPLAN-TASK MIL-CONDITION)
(status-of OPPLAN-TASK TASK-STATUS)
(specified-by OPPLAN-TASK OPPLAN)

STAFF-OFFICER: (*is-a-specialization* STAFF-OFFICER COMMANDER)
(is-a-planning-staff STAFF-OFFICER OPPLAN)
(is-an-adviser STAFF-OFFICER COMMANDER)

Fig. 2.6—OPPREFACE, OPPLAN-TASK, and STAFF-OFFICER structures

An OPPLAN-TASK is a specialization of an OPORDER-TASK. It inherits all of the relational structures of an OPORDER, and in addition it may have a condition, a MIL-CONDITION, and a status associated with it. The task will have the status, valid, in a world state only if the MIL-CONDITION is true in that world state. The restrictions on the OPPLAN, OPPREFACE, and OPPLAN-TASK structures are presented in the next subsection.

2.3.1. Restrictions on OPPLAN and its Components

OPP-RESN1: Every opplan is created-by the commander to whom the oporder-for the opplan was issued. This commander is also the assigned-comndr of the oporder-task-for the opplan:

[(created-by opplan (issued-to (oporder-for opplan))) AND
(created-by opplan (assigned-comndr-of (oporder-task-for opplan)))]

OPP-RESN2: The oporder-task-for an opplan should be one of the tasks specified by the oporder-for the opplan:

(oporder-task-for opplan (specifies-tasks (oporder-for opplan)))

OPP-RESN3: Every opplan is submitted to the commander who issued the oporder-for the opplan:

(submitted-to opplan (issued-by (oporder-for opplan)))

OPP-RESN4: An opplan may be approved by the commander to whom it was submitted-to:

(approved-by opplan (submitted-to opplan))

OPP-RESN5: An opplan may have the approved status, iff the commander to whom it was submitted has-approved it, and all the opplans-for the oporder-for the opplan have the approved status:

[(status-of opplan approved) IFF
([has-approved (submitted-to opplan) opplan)] AND
[(EVERY x (plans-for (oporder-for opplan)))
(status-of x approved)])]

OPP-RESN6: An opplan may have the active status only if it also has the approved status and in addition all the OPPLAN-TASKs specified by the opplan are valid:

[(status-of opplan active) IMPLIES
([status-of opplan approved] AND
[(EVERY x (specifies-tasks opplan))
(status-of x valid)])]

OPP-RESN7: The security classification of an opplan is higher than or the same as that of its associated oporder:

[NOT(is-lower (classification-of opplan) (classification-of (oporder-for opplan)))]

OPP-RESN8: The purpose of an opplan should be the same as the purpose-of the oporder-task-for opplan.

[purpose-of opplan (purpose-of (oporder-task-for opplan))]

OPP-RESN9: The objective of an opplan should be the same as the objective-of the oporder-task-for the plan:

[objective-of opplan (objective-of (oporder-task-for opplan))]

OPP-RESN10: The opplan is-mil-plan for the mil-action which the oporder-task-for the opplan specifies:

(is-mil-plan opplan (specifies (oporder-task-for opplan)))

OPPRF-RESN1: The security classification of an OPPREFACE should be the same or lower than that of its associated OPPLAN, but not lower than that of the oporder-for the OPPLAN:

((EXISTS opplan (subject-of oppreface))
 ([NOT (is-higher (classification-of oppreface)
                  (classification-of opplan))] AND
  [NOT (is-lower (classification-of oppreface)
                 (classification-of (oporder-for opplan)))]))

OPPTASK-RESN1: The objective-of a opplan-task is either the same as the objective-of the OPPLAN, opplan, in which the task is specified, or its objective supports the objective of opplan, or supports the objective of another OPPLAN-TASK, y, that is in (specifies-tasks opplan):

[(EXISTS opplan (specified-by opplan-task))
 ([objective-of opplan-task (objective-of opplan)] OR
  [supports (objective-of opplan-task) (objective-of opplan)] OR
  [(EXISTS y (specifies-tasks opplan)))
   (supports (objective-of opplan-task) (objective-of y))])]

OPPTASK-RESN2: Again, let opplan = (specified-by opplan-task). Then, the purpose of this opplan-task is governed by the following restriction:

(purpose-of opplan-task) = (purpose-of opplan) and
(objective-of opplan-task) = (objective-of opplan)

or
(purpose-of opplan-task) = (objective-of opplan) and
(supports (objective-of opplan-task)(objective-of opplan))
or there is another OPPLAN-TASK, y, such that
(purpose-of opplan-task (objective-of y)) and
(supports opplan-task y).

The logical statement of this restriction is shown below:

[(EXISTS opplan (specified-by opplan-task))
([(purpose-of opplan-task (purpose-of opplan)) AND
(objective-of opplan-task (objective-of opplan))] OR
[(purpose-of opplan-task (objective-of opplan)) AND
(supports (objective-of opplan-task) (objective-of opplan))] OR
[(EXISTS y (specifies-tasks opplan))
((supports opplan-task y) AND (purpose-of opplan-task (objective-of y)))])]

OPPTASK-RESN3: An opplan-task is valid iff the MIL-CONDITION specified by the condition-of the opplan-task is true:

[(status-of opplan-task valid)) IFF (ISTRUE (condition-of opplan-task))])]

2.3.2. The Function and Behavior of OPPLAN

The *function* of an opplan depends on its status. Fig. 2.5 shows the functions for three of its possible status values. If the status is completed, then this should *cause* it to be submitted-to the superior commander for approval. If the status is approved, then this should cause the oporder to be created from the opplan and distributed to all the COMMANDERs who are assigned to the OPPLAN-TASKs specified in the opplan, if any of the OPPLAN-TASKs has an assigned commander who is different from the one that created the opplan. If the status is active, then this should cause the execution-of the mil-actions and their respective *functions* to be achieved. The *behavior* of an opplan is simply the occurrence of its *function*.

It is appropriate to point out here the difference between ACHIEVE and CREATE operators, as well as the difference between OCCURS and ISTRUE.

2.4. CREATE, ACHIEVE, OCCURS, and ISTRUE Operators

The CREATE operator is used in Fig. 2.5 for the creation of the oporder from the opplan, and the ACHIEVE operator is used to specify the execution-of the mil-action. If the CREATE operator was successful, then it would cause the appropriate oporder to be created and installed as the one that the opplan gives-rise-to. The ACHIEVE operator would cause the processes that are needed to achieve the specified mila-execution to be started. If the mila-execution is NIL, then the ACHIEVE operator would trivially succeed by doing nothing. Instead of the ACHIEVE operator, if the CREATE operator had been used also for the (execution-of mil-action), then it would at best only cause the appropriate mila-execution to be created and installed as the value of (execution-of mil-action). It would not start the processes that are needed to perform the mila-execution. This is the distinction between ACHIEVE and CREATE. The formal statement of this distinction is presented in the operator inference rules specified in Appendix B.

Let us next consider the ISTRUE operator in the statement shown in OPPTASK-RESN3. There is no time parameter in this statement. Thus, the statement should be true in every world state, as per our convention mentioned earlier. The ISTRUE operator functions in the following environment: every object, concept and action, x, and every property associated with x (*via* a relation) in a world state, has a creation (and destruction) time associated with it. The creation (destruction) time of x (or its property) is the time it was created (destroyed). It is, of course, true that x can have a property at time, t, only if it had been itself created at t or before t. (ISTRUE (t proposition)) is true at the given time, t, in a world state, U_t, only if the proposition is true in U_t. Similarly, one has also ISFALSE and ISUNKNOWN operators. These operators should be contrasted with the OCCURS operator.

The OCCURS operator may be used either on particular instances of objects, concepts, and actions, or on expressions that use particular and generic instances and refer to processes (like CREATE, DESTROY, ACHIEVE, etc.), or on expressions that have truth values, but do not refer to processes. If it is used on an instance, x, then (t (OCCURS x)) = (OCCURS (t x)) will have truth value, T, if x exists in the world state, U_t. If it is used on a process description, say (t process-exp), then (t (OCCURS process-exp)) = (OCCURS (t process-exp)) will be true only if the processes referred to in the process-exp occur in the world state U_t. The truth value of such an OCCURS statement does not depend on the success or failure of the processes themselves. It may also have, of course, the truth value F, again depending only on process occurrences. If an expression does not refer to processes, then the truth value of (OCCURS expression) in a world state is the same as the truth value of expression itself in the same world state. The inference rules for reasoning about operators expressions is presented in Appendix B, and their use in solving a planning problem is discussed in Section 5.

2.5. The Concept of Building a Base

We have occasion to use the concept of building a base in the example discussed in Section 5. Let the name of this concept be BUILD-BASE. The principal *structure* of the BUILD-BASE action, the *function* of BUILD-BASE, and its *behavior* are briefly discussed here. Restrictions on the *structure* of BUILD-BASE are introduced in Section 5 as needed in the discussion of the example.

The tuple in Fig. 2.7 indicates the components of a BUILD-BASE action that gives it its unique identity, i.e., two build-base actions are different if and only if the BASE, REGION, or FORCE associated with them are different. The BASE is the base-of the action, the FORCE is the agent-of the action, and the REGION is the region-of the action. The location of the action will be the coordinate of the place where the action is taking place. This will be, of course, necessarily inside the region-of the action. The status-of this action is an ACTION-STATUS,* which can be dormant, active, completed, successful, failed, etc. Every action in CK-LOG has a starting time, an ending time, and a needed time specification. This is true also for the MIL-ACTION, even though it is not explicitly shown there. These are inherited from the *structure* of the ACTION concept. These are shown here explicitly because they will be used later in Section 5.

The needed-time-of a BUILD-BASE action is the time needed to complete the action successfully. This has been declared as being (WEEKS 2), a constant duration. When a constant is thus declared in the *structure* of a concept, this constant value is inherited, by default, by all the instances of the concept. Thus, all the build-base actions have thus been defined to require 2 weeks of time for successful completion. This is, of course, an oversimplification. In general, the needed time may depend on the type of base that is being built, where it is being built, and other features of the building process, and the base itself.

The type-of a BUILD-BASE action is the same as the type of the base that is being built; namely, a BASE-TYPE. The installations-in the action are BB-INSTALLATIONS. These will include all the installations

*Note that MIL-STATUS is a specialization of ACTION-STATUS. ACTION-STATUS is a general concept that appears with all actions in CK-LOG.

Structure: (BUILD-BASE BASE REGION FORCE)
(base-of BUILD-BASE BASE), 1
(agent-of BUILD-BASE FORCE), 1
(region-of BUILD-BASE REGION), 1
(location-of BUILD-BASE LOCATION), 1
(status-of BUILD-BASE ACTION-STATUS),
(starting-time-of BUILD-BASE TIME), 1
(ending-time-of BUILD-BASE TIME), 1
(needed-time-for BUILD-BASE (WEEKS 2)),
(type-of BUILD-BASE BASE-TYPE)
(installations-in BUILD-BASE BB-INSTALLATIONS)
(needed-resources-for BUILD-BASE BB-RESOURCES)
(defended-by BUILD-BASE FORCES)

Function: ([(during (*interval-of* build-base))
(PREVENT (DESTROY build-base))] AND
[(after (starting-time-of build-base))
(is-within (well-defended-region-of (agent-of build-base))
(region-of build-base))] AND
[ASSERT
((after (*time-of* (status-of build-base successful)))
([base-isin (base-of build-base) (region-of build-base)] AND
[location-of (base-of build-base) (location-of build-base)] AND,
[status-of (base-of build-base) operational] AND
[belongs-to (base-of build-base) (agent-of build-base)] AND
[type-of (base-of build-base) (type-of build-base)]))]

Behavior: ([(EXISTS x (needed-resources-for build-base))
((before (starting-time-of build-base))
(CREATE (location-of x (location-of build-base))))] AND
[(EXISTS x (installations-in build-base))
((after (starting-time-of build-base))(OCCUR (CREATE x)))]
AND
[(after (*time-of*
((EVERY x (installations-in build-base))(CREATE x))))
(ASSERT (status-of build-base successful))])

Fig. 2.7—The BUILD-BASE concept

of the type-of the base that is being built and may have some more that are a part of the building process itself. The needed-resources-for the action are the resources that are needed to build the installations of the base building operation. Finally, the base building operation itself may be defended by some FORCEs. These forces should, of course, belong to the agent of the base building operation.

The *function* of a base building action states what would happen when the base building operation is successfully completed, and that during its execution, its own destruction would be prevented. The base that is being built would come into existence at the place it is being built with the indicated properties after the successful completion of the operation. The region-of the base building operation will itself come into the well-defended region of the agent-of the action. The *function* of an action is invoked when the status of the action becomes active. The statements in the *function* of an action should always specify what becomes true in the world state when the action is successfully completed. In addition, it may also specify events that may occur during its active status (like the PREVENT statement above). The status of an action will become active only after the starting-time-of the action, when *behavior* of the action becomes true in the world state.

The *behavior* of BUILD-BASE states what happens during the building process: before the starting-time, there should exist at the location-of the build-base operation some of the resources needed to build the base. After the starting-time, the creation of the installations in the operation should occur. After every installation of the base has been built, the status-of build-base is asserted to be successful.

The statements in the *behavior* of an action are governed by the condition that the action is active if and only if its *behavior* is true. Thus, the standard way of destroying an action is by destroying its behavior. The use of the above behavior statement in the planning of the destruction of a base building operation is discussed in Section 5.

Let me conclude the definitions of the concepts presented so far with a brief sketch of a typical chain of activities that may occur in CK-LOG, when CK-LOG interprets the *function* statements as goals to be achieved in given world states.

2.6. The Use of Function Definitions as Goals

Suppose that in the world state, U_t, an OPORDER, x, was received by a group of commanders, and suppose that at some time after t, the user of CK-LOG typed in the input: '(ACHIEVE (*function* x)).' This will cause CK-LOG to set up for itself the *function* of OPORDER as a problem solving goal, for the particular OPORDER, x. By analyzing this goal (see *function* statement in Fig. A1 in Appendix A), it will now reduce this goal to a set of subgoals, one for each OPORDER-TASK specified in the oporder. This should cause CK-LOG to set up, for each commander that received x, the goal to create the task plan for his assigned OPORDER-TASK and have it approved. Let us now suppose that the user chose to focus on one of these subgoals, say, the one associated with the OPORDER-TASK, y. Let m be the commander assigned to this OPORDER-TASK. The subgoal will then be:

```
G: [ACHIEVE
     ((after (time-of (has-received m x)))
      (CREATE
       ([creates m (task-plan-for y)] AND
        [approved-by (task-plan-for y) (reports-to m)])))]
```

The commander, m, has already received x. So at this point when the user requests this subgoal to be achieved, CK-LOG will first identify the processes that are involved in the creation and approval of the OPPLAN, (task-plan-for y). The way CK-LOG does this using its general problem solving capabilities is discussed in Sections 3 and 5. This will, in effect, be partly specified by the OPPLAN *design* and

partly by the OPPLAN *function*. Suppose that with the help of the user, the design was successfully completed, and a new OPPLAN, z, thus got created as the task-plan-for y. The plan, z, will then have the completed status. When the *function* of z is now invoked (see *design* in Fig. 2.5), it will cause z to be submitted for approval. If it is approved, then, of course, the above subgoal, G, would have been achieved, and CK-LOG would proceed next to create the OPORDER for this plan z, as specified by the *function* for z. If it is not approved, then the subgoal G would fail. This may cause CK-LOG to revise the plan z, if the user so chooses, and resubmit it for approval. Thus, the conjunctive CREATE subgoal, G, can cause repeated creation and approval cycles for the plan z.

In this scenario, the user selected the goals that he wanted to explore with CK-LOG. One may also request CK-LOG to select subgoals automatically and proceed with their solutions. CK-LOG may use both the *function* and *behavior* of a concept to set up planning subgoals, and be guided in the goal setting and subgoal selection processes by the *analysis* and *design* statements associated with the concept. This is discussed in the context of the example in Section 5.

Before proceeding to the discussion of the planning example, it is useful to fix the structure of the language **DL** and the nature of the software system that interprets it. The reader has already encountered numerous examples of statements in this language. The concepts MIL-CONDITION and MIL-OBJECTIVE are statements in **DL**. Thus, typically a mil-objective might be,

'Prevent the establishment of a base by the enemy, Green, in Attu Island and during the period ...'

whose formal statement in **DL** might be,

```
[(during int)
 (PREVENT
  ((EXISTS base (bases-at Attu-Island))
   (belongs-to base Green)))],
```

where BASE is a concept in **U**, green is a FORCE, Attu-island is an ISLAND-REGION, a specialization of REGION, and 'PREVENT' is an operator. To be able to interpret statements like this and initiate the appropriate planning tasks, or to recognize the relationships between two different military objectives stated in this manner, the system should be able to analyze these statements and make the relevant inferences. The definition of the concept of MIL-OBJECTIVE will characterize the *structure* of the statements in **DL** that may appear as military objectives and specify their *function*, *analysis*, and *design*. CK-LOG will use this definition to control the inferences it does on a given mil-objective.

Similarly, the concepts MIL-EXECUTION and MIL-SITUATION-DESN will involve statements in **DL** that refer to actions and objects in the operational planning universe **U**. To adequately describe the universe **U** to the system, it is thus necessary to define not only the concepts in the universe **U**, but also concepts of statements like the ones above and inference rules for them.

The inference rules fall into two categories: the *Logical Rules* and the operator rules. The logical rules are common to all domains. They are dictated by notions of consistency and completeness of the general logical system that is used here. The logical system of inference and the inference rules used in CK-LOG were first formulated by the German logician Gentzen [8] and later developed by Kanger [2]. Examples illustrating the use of this logical system are presented in Section 3. The logical inference rules used by this system are presented both in Section 3 and Appendix B. These inference rules provide the basis for stating and solving problems in CK-LOG. The operator rules specify methods for reasoning with the various logical operators that we have used in **TML**. These are presented in Appendix B. The system architecture of CK-LOG that enables it to use the inference rules to analyze and solve problems is discussed in both Sections 3 and 5.

3. THE CK-LOG SYSTEM

The previous section presented numerous examples of statements in TML and DL, the kinds of knowledge described in CK-LOG, and discussed informally their meaning and use in CK-LOG. The way CK-LOG understands statements in these languages is critically dependent upon its architecture: its own component subsystems, their functions and behaviors, the way knowledge described to CK-LOG is interpreted and used, and CK-LOG's own notions of problems and solutions. These are explained in this section.

3.1. CK-LOG's Knowledge of a Universe

3.1.1. Structures and their Interpretations

The knowledge of a universe is viewed in CK-LOG as being a set of *concepts*. Concepts are organized in a specialization and generalization hierarchy, where the specializations inherit the properties* of their generalizations by default, and the specializations may have their own additional properties. In the terminology used in the literature of *knowledge representation systems*, the knowledge of a universe, U, which we denote here by K[U], is usually called the *domain knowledge*, and the universe, U, is called the *domain of discourse*.

The *structure* of a concept defines the relational forms that are used to describe the relationships between the concept and other concepts in the universe. The relational forms

(relation-name CONCEPT-1 CONCEPT-2) and
(tuple-name CONCEPT-1 ... CONCEPT-k)

are called *dimensions* in CK-LOG. In choosing this name, I am suggesting an analogy between forms like

(parent-of INFANT PERSON),
(commander-of MIL-ACTION COMMANDER), ...,

and forms like

(weight-of OBJECT POUNDS),
(length-of OBJECT FEET),
(area-of REGION SQUARE-MILES), ...,

which are normally viewed as specifying the dimensions of the indicated measurements. It is dimensionally inconsistent to say, '(length-of OBJECT POUNDS)' or '(area-of REGION FEET).' Similarly, one might say that forms like '(parent-of INFANT MIL-ACTION),' '(commander-of MIL-ACTION INFANT),' etc., are also dimensionally inconsistent in our universe.

A dimension, '(r X Y),' is interpreted in CK-LOG as specifying that there are world states in the universe in which there exist instances x of the concept X, and y of the concept Y, for which '(r x y)' is true:

((r X Y) IFF
[(EXISTS U_t WORLD-STATE)(EXISTS x X)(EXISTS y Y)
(ISTRUE (r x y) U_t)])

*Properties of concept refers here to the *structure*, *function*, *behavior*, *design*, and *analysis* aspects of a concept.

Thus, if '(r X Y)' has not been declared as a dimension, then in every world state of the universe '(r x y)' will be false, for all instances x of X and y of Y. If '(r X Y)' is a dimension, then we shall say that '(r x y)' is an instance of '(r X Y)' if x is an instance of X, and y is an instance of Y. Clearly, not every instance of a dimension will be true in a world state. For a given instance x of X and given relation name, r, the *restrictions* associated with '(r X Y)' will specify the conditions under which '(r x y)' may be true in a world state, for a particular instance y of Y.

Thus, the restrictions on '(specified-by MIL-ACTION OPORDER-TASK)' state that a MIL-ACTION is always specified by an OPORDER-TASK, which is one of the tasks of the OPORDER that ordered the MIL-ACTION. Also that, every MIL-ACTION is uniquely specified by only one OPPORDER-TASK, and every OPORDER-TASK specifies only one MIL-ACTION. These restrictions should be satisfied in every world state; i.e., the conditions that specify the restrictions should not be false in a world state, for any MIL-ACTION that is in the world state. The truth value associated with these conditions, however, may be ? (unknown), because of insufficient information. In any case, one does not have the freedom to arbitrarily assign an OPPORDER-TASK as the one that specified a given MIL-ACTION. The interpretation of restrictions like these in the world states modelled by CK-LOG is summarized by the following:

> Every restriction associated with every dimension should evaluate to either T or ? in every world state, for all instances of the dimension that appear in the world state.

The representation of this world state in CK-LOG is called the *model.* A model (i.e., world state representation) is said to be *contradiction-free* if and only if none of the restrictions associated with the dimensions in **K[U]** evaluate to F (i.e., is false) in the model. A model is said to be *consistent* if every restriction, associated with the instances in the model, evaluate to T in the model. Thus, formal statements of these restrictions in CK-LOG are called *consistency conditions.* CK-LOG has special facilities for building and updating models that are contradiction-free with respect to a given set of consistency conditions. The subsystem of CK-LOG that does this is called the Meta Description System (MDS) [3-5, 9, 10].

3.1.2. Operators and Actions in CK-LOG

In our universe, ACTIONS are used to create, destroy, or to keep objects, and to create (make true), destroy (make false), or keep (maintain the truth or falsity of) relationships between objects in a world state. Thus, as mentioned before, every concept and dimension in CK-LOG will have three kinds of actions associated with them: the **create-action** for the creation of an instance of the concept or dimension, the **destroy-action** for its destruction, and the **keep-action** to maintain it in the world state. Section 5 gives examples of the create and destroy action specifications.

As we have seen in Section 2, operator expressions are used in **DL** to implicitly refer to the actions that are needed to create, destroy, or maintain these entities in a world state. Thus, (CREATE opplan) will implicitly refer to whole sequence of actions that are needed to create the opplan. CK-LOG will identify these actions from the *design* specification of OPPLAN (discussed in Section 5). Similarly, (CREATE (commander-of mil-action_i comndr_i)) will implicitly refer to the actions needed to make comndr_i the commander of mil-action_i; namely, that there should exist an OPORDER-TASK that is a task of the OPORDER that orders the mil-action_i, such that mil-action_i is the mil-action-of this OPORDER-TASK and comndr_i is its assigned commander. The logical statement of this is shown below:

(CREATE
 [(EXISTS oporder (ordered-by mil-action_i))
  (EXISTS oporder-task (specifies-tasks oporder))
  ((mil-action-of oporder-task mil-action_i) AND
   (assigned-comndr-of oporder-task comndr_i))])

This might be the **create-action** for the (commander-of MIL-ACTION COMMANDER) dimension. To do (CREATE (commander-of mil-action_1 comndr_1)), CK-LOG will initiate the actions needed to make the above expression true in the world state. If the actions are successful, then it will result in making comndr_1 the commander of mil-action_1. If such an action definition is not available, then CK-LOG will use the logical restriction associated with this dimension to deduce expressions like the one above. If no logical restriction exist for the dimension, then it will simply assert the relation into the world state.

When the argument of the CREATE expression is a complex logical expression like the one above, the *inference engine* of CK-LOG will analyze this expression, breaking it down to its component elementary parts, and identify the actions needed to make the expression true in a world state by putting together, in the right manner, the actions necessary for making the component parts true. Section 5 gives several examples of this kind of reasoning by CK-LOG.

The possible outcome of a successful execution of an action in a world state is described by the *function* associated with the action. Thus, the *function* of a MIL-ACTION specifies its possible outcome; namely, that its objective will be realized. The particular changes that occur in a world state as a result of the realization of the objective of a military action will, of course, depend on the objective itself. Again, by analyzing this objective using its *inference engine*, CK-LOG can identify these changes.

The *behavior* of an action describes what happens in a world state while the action is occurring in that world state. We have seen examples of this in Section 2 for MIL-ACTION and BUILD-BASE. The behavior of an action is used in CK-LOG for two purposes: to keep track of the progress of an action in a world state and to reason about it, or to destroy an action in the world state. Examples of these kinds of processing in operational planning appear in Section 5.

Similarly, the consequences of introducing an instance of a concept into a world state are also specified by the *function* associated with the concept. Thus, the *functions* of OPORDER and OPPLAN specify what happens when they are created.

Once **K**[U] is defined for a universe, U, CK-LOG will have in its knowledge base the knowledge necessary to build models of world states in the universe, to check the consistency of these models, to invoke the actions needed to change the models, and to reason about the consequences of the actions. As we have seen above, this knowledge is quite diverse and itself has a rather complex structure. Also, for a universe like the Naval Operational Planning universe, this knowledge base will be quite large. To use this knowledge effectively, the system should have the capability to identify the pieces of knowledge that are relevant to a given problem and bring this knowledge together in an organized manner to the solution of the problem.

It is inconceivable that one could specify to CK-LOG all the possible ways in which knowledge of this kind might be used. It would be convenient, if the system had the capability to analyze a given problem, to identify the pieces of information necessary to solve the problem, and use this information automatically in the right manner to solve the problem. This is, however, an almost impossible task. CK-LOG offers the next best alternative: it can use **K**[U] automatically to analyze a problem, identify the information needed to solve it, reason about it, and present its reasoning to a user. But in most cases, it can solve the problem only with the help of the user. To understand how CK-LOG does this, it is necessary to understand how problems are stated, and what it means in CK-LOG to solve a problem.

Problems and inference rules are stated in CK-LOG in a sublanguage of **TML**, called the language of *sequents*. The language of sequents is a generalization of the language of *Horn Clauses* used in PROLOG [11]. The logical calculus used in PROLOG's inference engine is based on the *resolution theory* [12]. The logical calculus used by CK-LOG's inference engine is called the *calculus of sequents* [2]. The theory associated with this calculus is called the *theory of natural deduction* [8]. CK-LOG uses a

modified version of this calculus of sequents. The modifications are designed to achieve five purposes: first, to make use of the models of world states in the reasoning processes; second, to access from the knowledge base the pieces of knowledge that are relevant to a given problem; third, to identify information in the world state that is relevant to the solution of a problem but is not known in the world state; fourth, to update the world state based on findings from the problem solution; and fifth, to reason about operator expressions that appear in the problem statements.

Just as PROLOG is a logic-programming system based on resolution theory, CK-LOG is a logic-knowledge-processing system based on the theory of natural deduction. The logical theory of natural deduction is not presented here, nor is there an attempt to establish the validity of the variant of the calculus of sequents that is used in CK-LOG. Descriptions of *sequents* and *inference rules* are presented later in this section, and the way the inference engine of CK-LOG uses them, to state and solve problems, is explained. The principal difference between CK-LOG and knowledge representation systems is discussed in the next subsection.

3.1.3. CK-LOG as a Knowledge Processing System

The architecture of the CK-LOG system is shown in Fig. 3.3 and is discussed in Section 3.5. The principal contribution that this architecture makes to the knowledge representation systems technology is that it shows a way of not only representing the knowledge **K[U]**, but also using this knowledge automatically in the appropriate manner to help a user state and solve problems. In this sense, CK-LOG is not just a knowledge representation system, but it is a *knowledge processing system.*

This feature makes CK-LOG significantly different from the *frame-based* knowledge representation systems like KRL [13], FRL [14], KL-ONE [15], UNITS [16], and others. As the reader may have noticed, the *structure* of a CK-LOG concept is syntactically quite similar to the structural units of knowledge called, *frames*, that are used in the frame-based knowledge representation systems. They both provide a way of organizing the knowledge of a universe into a taxonomic hierarchy of specializations and generalizations, with rules for inheritance of properties and functions over the hierarchy. Each taxonomic unit (like COMMANDER, FORCE, etc.) corresponds to a class of objects in the universe. The *frame* definitions (along with the *structure* definitions in CK-LOG) specify to a computer how these classes of objects are represented in the computer. They also specify a domain language to describe the objects and state problems.

The knowledge engineer who specifies the *frames* should also write the programs that inform the computer how it might use the frames to model world states, and how it might use its model building capability to solve a set of problems. This model building capability is the basis for all problem solving. The *frame* structures acquire a meaning only with respect to this capability. There is no notion of *logical semantics*, i.e., meaning that is well-defined with respect to a general logical deduction system. In short, the meaning of sentences in the domain language, and the set of problems that a *frame-based* system can solve, will both depend on the programs written by the knowledge engineer.

This is the situation that should be contrasted with the situation in CK-LOG. The knowledge engineer who defines **K[U]** to CK-LOG does not have to write such programs to tell CK-LOG how it should use this knowledge. This is because, unlike the *frames*, the *structure* of a CK-LOG concept, as also its *function, behavior, design*, and *analysis*, have a logical semantics. Thus, CK-LOG is able to use a general logical deduction system in its inference engine to solve problems. This inference engine provides the basis for all of its problem solving and model-building activities. Just as *horn clause* statements acquire a *procedural interpretation* in the context of PROLOG's inference engine, *sequents* acquire a procedural interpretation in the context of CK-LOG's inference engine. The nature of this inference engine is briefly discussed in the next subsection.

3.2. CK-LOG's Inference Engine

3.2.1. Sequents, Problems, and Theorems

A *sequent* has the form,

[S1]: $X_1, ..., X_n \rightarrow Y_1, ..., Y_m$; $0 \leqslant n$, and $0 \leqslant m$,

where the X's and the Y's are arbitrary logical expressions in TML. One may read this sequent as,

from (X_1 and X_2 and ... and X_n), conclude (Y_1 or Y_2 or ... or Y_m).

The logical expressions on the left side of $\rightarrow$ may be viewed as known facts, or hypotheses. The sequent says, given that the conjunction of known facts or hypotheses is true, one may conclude the disjunction of the expressions on the right side of $\rightarrow$.

A sequent is said to be *valid* iff

[L1]: [(X_1 AND ... AND X_n) IMPLIES (Y_1 OR ... OR Y_m)]

is a *theorem*. This means that in every model (world state) in which the antecedent is true, the consequent is also true. One may think of this as saying that a sequent is valid iff the conclusion proposed in the sequent is provably correct. A bullet is placed instead of ';' in a sequent to indicate that the sequent is valid as in,

[T1]: $X_1, ..., X_n \rightarrow Y_1, ..., Y_m$ •

A sequent without this bullet is interpreted as a problem to be solved. The problem is, of course, to prove that the sequent is valid. The problem,

[P1]: $\rightarrow Y_1, ..., Y_m$;

with an empty left side is interpreted in CK-LOG as,

given **K**[U] and the current U_t, prove that
one may conclude (Y_1 OR ... OR Y_m).

The problem,

[P2]: $X_1, ..., X_n \rightarrow Y_1, ..., Y_m$;

is interpreted as,

assuming the hypotheses X_1, ... and X_n, and given **K**[U] and U_t, prove that one may conclude (Y_1 OR ... OR Y_m).

Thus, in CK-LOG the logical statements corresponding to **K**[U] and the facts in U_t are assumed to be always available on the left side of each problem. When a user types:

$\rightarrow$ (ACHIEVE (*design* opplan)) ;

to CK-LOG, the user is stating a problem to CK-LOG. The problem here is to achieve the design of the operational plan, opplan. Here the specifications for the design of the opplan are assumed to be

already in the current world state. As shown in Section 5, CK-LOG will use its inference engine and its model building capability to interpret the above problem, to identify the knowledge in **K[U]** that is relevant to the problem, translate it to the appropriate logical expressions in **DL**, to place them in the problem sequent when needed, and proceed in its attempt to solve the problem. To understand this process, it is first necessary to know what the concept of a proof is in the language of sequents, and how proofs are constructed. This is discussed in the next subsection.

3.2.2. The Concept of a Proof

In presenting the concept of a proof and the calculus of sequents, to simplify the discussion, assume that **K[U]** and U_t are empty, and for each problem all the needed facts and hypotheses are explicitly given on the left side of the problem sequent. The calculus of sequents assumes a single axiom. The axiom is,

$$X \rightarrow X \bullet;$$

namely, that 'from X one may conclude X' for any logical expression X. The general statement of this axiom has the form,

$$[A]: \ .., X, ... \rightarrow ...., X, ... \bullet$$

where the dots indicate that the expression X may be surrounded on both the left and the right side of $\rightarrow$ by an arbitrary number of (zero or more) other logical expressions. The differing number of dots indicate that the expressions in the various locations need not all be the same. Clearly,

$$[(.. \text{ AND } X \text{ AND } ...) \text{ IMPLIES } (.... \text{ OR } X \text{ OR } .....)]$$

is a theorem. Thus, axiom [A] is valid.

The calculus of sequents provides rules for transforming a problem sequent to one or more simpler problem sequents, preserving the validity of the sequent in this process; i.e., if the original sequent is valid, then all the transformed sequents are also valid, and if the original sequent is not valid, then at least one of the transformed sequents is not valid. Through successive transformations if one is able to reduce a problem sequent to the form of the axiom [A] (or to a set of sequents each having the form of [A]) then, since all transformations preserve validity, the problem sequent should also be valid, and this would constitute the proof of the problem. This process is illustrated in Fig. 3.1, where '[A]' in the leaves of the tree indicate that the sequent at that place is an axiom, and '[P]' indicates that the sequent at that place is a problem. If one is not able to reduce the problem sequent at the root of this tree to axioms in this manner, then the problem sequent is not valid. The tree of deductions shown in Fig. 3.1 is called *proof tree* or *deduction tree*. The set of problem sequents in the leaf nodes of a deduction tree is called the *frontier set* of the deduction tree.

The set of rules used in the calculus is *complete* in the sense that if one started with a valid sequent at the root, then one is guaranteed termination with axioms in all leaf nodes of the proof tree. It is *consistent* in the sense that if one found a proof tree with axioms in all leaf nodes, then the problem at the root is indeed valid. The rules for the sequent calculus are presented in Appendix B. The calculus has four stages of development:

1. The calculus for propositional expressions (i.e., expressions that do not contain quantifiers, EVERY or EXISTS, or operators).

2. The calculus for quantified expressions (i.e., expressions that contain expressions of the form '(EVERY ..)' and '(EXISTS ..)'). The rules for this calculus are presented in Appendix B.

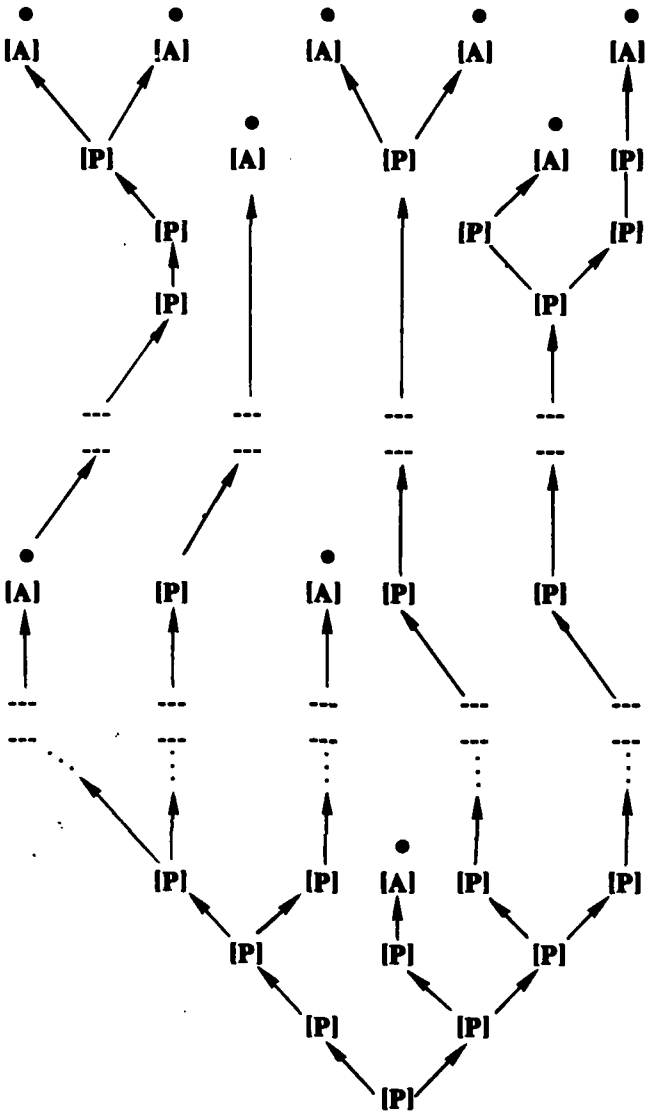


Fig. 3.1—The structure of a proof tree in the calculus of sequents

3. Special facilities introduced in the calculus in CK-LOG to communicate with the knowledge base and the world states.

4. Specialized inference rules used in the calculus to analyze operator expressions and timed expressions. These rules are also presented in Appendix B.

The calculus for propositional expressions is discussed in the next section in some detail. Understanding this calculus and the way it is used to generate proofs is basic to the understanding of CK-LOG's problem solving methods. After introducing the calculus for propositional expressions, some simple examples are discussed to illustrate how the calculus works for quantified expressions (expressions containing 'EVERY' and 'EXISTS' terms) and operator expressions (expressions containing ASSERT, CREATE, DESTROY, etc. operator terms), and how the calculus is used to reason about actions.

3.2.3. The Propositional Calculus of Sequents

The transformation rules are shown in Table 3.1. Each rule has two parts: a bottom sequent and a top sequent. It is to be interpreted as follows: any time a problem sequent is found that matches the pattern shown in the bottom sequent, it may be transformed to the sequent(s) shown on the top. The dots again indicate that the patterns shown may be surrounded by an arbitrary number of other expressions in the sequent, separated by commas.

Table 3.1—Propositional Rules

Rule Name	Inference
[AND →]	..., x, y, .. → .... ;
	..., (x AND y), .. → .... ;
[→ AND]	.... → ..., x, .. ; | .... → ..., y, .. ;
	.... → ..., (x AND y), .. ;
[→ OR]	.... → ..., x, y, .. ;
	.... → ..., (x OR y), .. ;
[OR →]	..., x, .. → .... ; | ..., y, .. → .... ;
	..., (x OR y), .. → .... ;
[→ N]	...., x → ..., .. ;
	.... → ..., (NOT x), .. ;
[N →]	..., .. → ...., x ;
	..., (NOT x), .. → .... ;
[→ IMP]	...., x → ..., y, .. ;
	.... → ..., (x IMPLIES y), .. ;
[IMP →]	..., y, .. → .... ; | ..., .. → ...., x ;
	..., (x IMPLIES y), .. → .... ;

The first rule in the table is the *left AND elimination* rule, called [AND →], for short. It is used to eliminate the AND connective that appears on the left side of a sequent. It says in effect that any time an AND expression, (x AND y), is found on the left side of a sequent, separated by commas from other expressions, then this AND may be eliminated by using the transformed sequent shown on top, where the AND is replaced by commas keeping all the rest unchanged.* Clearly, this is consistent with our interpretation of sequents shown in [L1] in Section 3.1.1: the commas on the left side do stand for AND's. Thus, if the bottom sequent is valid, then so is the top one; and if the bottom sequent is not valid, then so is the top one. Therefore, this rule does preserve validity. The dual of this rule is the *right OR elimination* rule, [→ OR]. Here, OR is eliminated by commas.

*In DL one may have expressions of the form (x1 AND x2 ... AND xn). In CK-LOG the [AND →] rule shown in Table 3.1 is extended to cover this case. To simplify the presentation, I have not shown the general case in this table.

The *right AND elimination* rule, [→ AND], may be interpreted as follows: if there is a problem in which one wants to prove (x AND y), then this problem may be split into two problems, one to prove x, and the other to prove y, keeping all the rest in the problem sequent unchanged. If these two smaller problems are both proven valid, then clearly the original problem is valid. If at least one of the two smaller problems is not valid, then the original problem is also not valid. Thus, this rule also preserves validity.

The dual of this rule is the *left OR elimination* rule, [OR →]. Here also the problem sequent is split into two subproblems. This may be interpreted as follows: a conclusion from (x OR y) is valid if and only if the conclusion from x alone is valid, and the conclusion from y alone is also valid, keeping all the rest unchanged. Notice that (x OR y) can be true in three ways: when x is true and y is false, y is true and x is false, and both x and y are true. To prove the conclusion for (x OR y), one should thus be able to prove it in all the above three cases. All these three cases are covered by the two subproblems generated by the [OR →] rule.

The validity of the negation rules [→ N] and [→ N] are not quite as obvious as the validity of the above rules. They say that a negated expression on one side of a sequent may just be moved to the other side after removing the negation, and this would preserve the validity of the transformation. To see why this is true, consider the [→ N] rule.

Suppose the bottom sequent in [→ N] was valid. Then, the conjunction of the expressions on the left side (call this C1) implies the disjunction of the expressions (call this (D1 OR (NOT x))) on the right in all world states, i.e.,

[1]: [C1 IMPLIES (D1 OR (NOT x))].

Since the transformation preserves validity, it follows that

[2]: [(C1 AND x) IMPLIES D1]

should also be true in all the world states. Let U_t be any arbitrary world state. In this world state, if C1 is false, then both the implications [1] and [2] above are trivially true in this world state.* If C1 is true then, since the sequent is valid, (D1 OR (NOT x) should be also true, since the implication [1] is true. There are two cases to consider here: in one case (NOT x) is true in U_t, and in the other (NOT x) is false. If (NOT x) is true, then x is false, and in this case the implication [2] is trivially true. If (NOT x) is false, then x and D1 are both true, and this makes the implication [2] true. Thus, in every world state, the implication [2] is also true. Hence, the top sequent in the rule is also valid.

If the bottom sequent in [→ N] is not valid, then there is a world state in which the left side of the implication [1] is true and the right side is false. In this case, both D1 and (NOT x) are false. This makes the implication [2] also false in the same world state. Hence, the top sequent of [→ N] is also not valid. Thus, the rule preserves validity. By a similar argumentation, one can show that the rule [N →] also preserves validity.

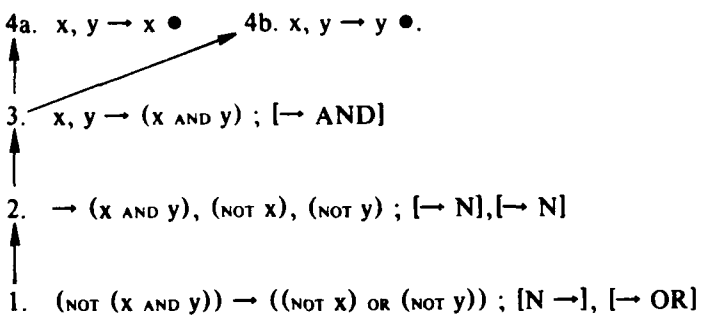
The [→ IMP] rule may be derived by transforming the implication (x IMPLIES y) to ((NOT x) OR y) and applying the rules ([→ OR], [→ N]) in sequence to the resulting sequent. Similarly, the [IMP →] rule is obtained by applying ([OR →], [N →]) to the transformed implication. One may similarly construct also the [IFF →] and [→ IFF] rules. They are not shown in Table 3.1.

All the rules shown in Table 3.1 preserve the validity of the sequents. Also note that each rule eliminates the occurrence of a logical connective from a sequent. After the application of any of these

*(x IMPLIES y) is logically equivalent to ((NOT x) OR y), and if x is false then ((NOT x) OR y) is clearly true, whether y is true or not.

rules, the resultant sequent(s) will be simpler in the sense that they will contain one less logical connective. Since any finite propositional problem will contain only a finite number of connectives in it, repeated applications of these rules to a problem should thus ultimately result in sequents that contain no logical connectives whatsoever. After this point is reached, no more rules could be applied to any of the sequents. At this point one may test whether each of these terminal sequents is an axiom or not. If all of them are axioms, then the initial problem is valid. If any of the terminal sequents is not an axiom, then the initial problem is not valid. The simple examples shown in Fig. 3.2 illustrate this process. The rules applied to each problem sequent are shown in this figure to the right of the sequent in the order they are applied. The terminal sequents are both axioms in the first problem. Thus, the root problem is valid. The terminal sequents 2a and 3 in the second problem are both not axioms. Here the root problem is not valid. The rules for reducing quantified expressions are presented next.

PROOF TREE 1:



PROOF TREE 2:

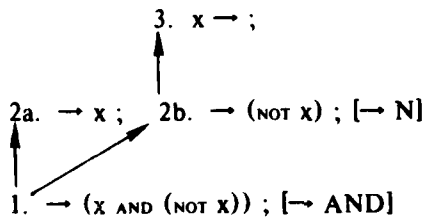


Fig. 3.2—Examples of proofs in propositional logic

If problem sequents contain quantified expressions and operator expressions, then the inference rules for quantified expressions and operator expressions are used to transform these expressions into their equivalent propositional forms, and then an attempt is made to reduce these transformed sequents to axioms using the propositional rules discussed above. To illustrate the principles involved in this process with simple examples, consider, for example, the very simple problem,

→ [((EVERY x)(loves x x)) IMPLIES ((EVERY x)(EXISTS y)(r x y))] ;

Here, we want to prove the following: if every person loves himself, then for every person, x, there is a person y such that x loves y. This has a rather simple proof: let p be any person. Then (loves p p) is true. Hence, there is a person who loves p. The proof in natural deduction is quite similar to this. It proceeds as follows: the application of [IMP →] rule to the above problem will transform it to,

[(EVERY x)(loves x x)] → [(EVERY x)(EXISTS y)(loves x y)] ;

The quantifier elimination rule is now applied to the above problem to eliminate the 'EVERY' quantifer. This will result in the following sequent:

$$(\text{loves uGx uGx}) \rightarrow ((\text{EXISTS y})(\text{loves p y}))\ ;$$

where uGx is a new variable (called the *universal generalization variable*), p is a new constant (called the *universal instantiation constant*), both of which are generated during the rule application. The elimination of 'EVERY' quantifiers on the left of '→' will produce new generalization variables, and their elimination on the right of '→' will produce new instantiation constants. The constant 'p' created above is the analog of the statement 'Let p be any person' in the informal proof presented earlier. At this point, the rule to eliminate the 'EXISTS' quantifier is applied to produce the new problem,

$$(\text{loves uGx uGx}) \rightarrow (\text{loves p eGy})\ ;$$

where eGy is again a new variable (called the *existential generalization variable*) generated by rule application. Now, if the variables uGx and eGy are bound to p, then for this binding the above sequent reduces to an axiom and the proof is done. This is indicated in the proof by the axiom,

$$(\text{loves p p}),(\text{loves uGx uGx}) \rightarrow (\text{loves p eGy}),\ (\text{loves p p})\ \bullet$$

The test used to check whether a sequent is an axiom or not is called the *axiom test.* If an axiom test is successful for a sequent, Q, in a leaf node of a deduction tree, then of course Q will contain an identical pair of atoms on either side of Q. If every sequent in the frontier set of a deduction tree thus reduces to an axiom, then the proof is done, and the root problem is valid.

As shown in the example above, if a problem sequent, P, contained quantified expressions, then at the time an axiom test is performed, the atoms in the transformed and reduced version of P, say Q, may contain variables. One has to bind these variables to constants before performing the axiom test. A general rule here is that a variable, x, may be bound to a constant, c, only if the level in the deduction tree at which c was generated was not higher than the level at which the variable x was generated. In the above case, the variable uGx and the constant p were both generated at the same level, the first level, of the deduction tree.

Proof generation is in general not this simple. The number of possible bindings for variables in a sequent can quickly grow astronomically large, so a systematic search procedure is necessary. In CK-LOG this search procedure is given by the proof termination algorithm called the *mating algorithm.* The details of this algorithm are discussed in Ref. 1.

Here is a simple example that illustrates the way CK-LOG reasons with operator expressions, and interacts with the world state and the knowledge base during the proof construction process. Suppose we wanted to prove the following:

$$\begin{aligned}[\text{P1}]:\ \rightarrow\ &((\text{EVERY x FORCE})(\text{EVERY y FORCE})(\text{EVERY r REGION})\\ &[(\text{MIL-ACTION x y r})\ \text{IMPLIES}\\ &((\text{EXISTS z FORCE})\\ &[(\text{belongs-to z x})\ \text{AND}\ (\text{OCCURS}\ (\text{DESTROY z}))])])\ ;\end{aligned}$$

i.e., in every military action prosecuted by x, there will be forces, z, belonging to x against which DESTROY operations will occur. This is a statement that is true in all world states, as per *behavior* of military action given in Fig. 2.1. To prove statements like this, i.e., ones that are true in all world states, one would start the problem with the empty world state associated with it. Thus, the world state associated with [P1] will be empty. In this case, the application of 'EVERY' and 'EXISTS' elimination rules will transform this problem to:

[P2]: → [(MIL-ACTION f1 f2 r) IMPLIES
((EXISTS z FORCE [(belongs-to z f1) AND (OCCURS (DESTROY z))]) ;*

This transformed sequent here has generated a particular military action between forces f1 and f2 in region r. This corresponds to the statement, 'Let (MIL-ACTION f1 f2 r) be any military action.' Whenever a new instance of an action is thus created, this action will be introduced into the world state associated with the problem sequent. Thus, in the world state associated with the above sequent, forces f1 and f2 and region r will be created and a MIL-ACTION with these forces and the region will be established. Let us call this MIL-ACTION, ma. Since no time specifications were made here, the time associated with ma will be the *current time* in the world state, i.e., some arbitrary time, t. This will now cause the system to retrieve from the knowledge base the *function* and *behavior* of MIL-ACTION, specialize these to ma, and introduce these specializations on the left hand side of the problem sequent, as shown below:

[P3]: [(before (*time-of* (status-of ma successful))) (**value-of** (*function* ma))],
[(before (**value-of** (*function* ma))) (**value-of** (*behavior* mil-action))]
→ [(MIL-ACTION f1 f2 r) IMPLIES
((EXISTS z FORCE [(belongs-to z f1) AND (OCCURS (DESTROY z))]) ;

The function **value-of** is used to retrieve from the knowledge base the definitions of *function* and *behavior* of MIL-ACTION and substitute the constant ma for the variable mil-action used in these definitions. In effect, the system is now hypothesizing that before the military action ma terminates successfully, its *function* would be true, and before its function becomes true, its behavior would be true. The expressions on the left side of the sequent above describe what will happen to the world state as a result of ma. Expressions of the form '[(before (*time-of* **event**)) **exp**]' are called event-time expressions. When a reduction rule is applied to an expression like this, it will postulate a time, say elt1, at which the event becomes true, and note

elt1 = (*time-of* **event**).

Then it will transform the expression in the sequent to,

[**event** IMPLIES ((before elt1) **exp**)],

which in fact is precisely what we mean by the event-time expression above. Application of this rule to our problem sequent above will transform it to the following:

[P4]: [(status-of ma successful) IMPLIES ((before elt1) (**value-of** (*function* ma)))],
[(**value-of** (*function* ma)) IMPLIES ((before elt2) (**value-of** (*behavior* mil-action)))]
→ [(MIL-ACTION f1 f2 r) IMPLIES
((EXISTS z FORCE [(belongs-to z f1) AND (OCCURS (DESTROY z))] ;

(elt1 = (*time-of* (status-of ma successful))) and
(elt2 = (*time-of* (**value-of** (*function* ma)))).

*I am not presenting here all the steps in the proof of [P1]. My intent here is only to illustrate the essential nature of the proof generation process, and it is not to establish its correctness. For the special cases that occur in the problems presented above, I have made some simplifying assumptions on the application of the quantifier elimination rules. The discussion in Section 5 presents the steps in proof generation in greater detail.

Let **success** denote '(status-of ma successful),' **fn** denote '(**value-of** (*function* ma)),' **beh** denote '(**value-of** (*behavior* mil-action)),' and **rhs** denote the right side of [P4]; namely, '[(MIL-ACTION f1 f2 r) IMPLIES ((EXISTS z FORCE [(belongs-to z f1) AND (OCCURS (DESTROY z))].' Then, [P4] may be rewritten as,

[P4]: [**success** IMPLIES ((before elt1) **fn**)],
[**fn** IMPLIES ((before elt2) **beh**)] → **rhs** .

Applying [IMP →] twice to the implications on the left side of [P4] results in,

[P5a]: **fn**, ((before elt2) **beh**) → **rhs** ;
[P5b]: ((before elt1) **fn**) → **fn** •
[P5c]: ((before elt2) **beh**) → **success** ;
[P5d]: → **success**, **fn** .

In [P5b] above, **fn** will match with '((before elt1) **fn**)' and thus reduce it to an axiom. All the above sequents have the same world state associated with them. The proof of [P5a] will cause this world state to be modified according to **beh**. The expansion for **beh**, taken from Fig. 2.1 with mil-action substituted by ma, is shown below:

beh: [(EXISTS int INTERVAL)
[((during int)(MIL-ACTION f1 f2 r)) IMPLIES
([(EXISTS force FORCE)
((between int)
([OCCURS (DESTROY force)] AND
[(belongs-to force f1) OR (belongs-to force f2)]))]
AND
[(during int)(OCCURS (ACHIEVE (execution-of ma)))]

AND
[(EXISTS r1,r2 (is-within r))
((between int)((controls f1 r1) AND (controls f2 r2)))]))]]

(I have boxed the last AND above because I will be referring to this AND later below.) Let me call the last conjunct,

[(EXISTS r1,r2 (is-within r))
((between int) ((controls f1 r1) AND (controls f2 r2)))])

by the name **rcontrol**. The above expression for **beh** will now take the place of **beh** in sequent [P5a]. The following sequence of rules are now applied to [P5a]:

First, the [→ IMP] rule is applied to eliminate the IMPLIES connective from **rhs** in [P5a]. This will cause '(MIL-ACTION f1 f2 r)' to move to the left side of the sequent. Then the 'EXISTS' quantifier in **rhs** is eliminated. This will generate a new variable, eGforce, which is substituted for the variable, z, in **rhs**. After this, the analysis of '((before elt2) **beh**)' using the appropriate rules will result in:

[P6a] **fn**, [(between int1)(OCCURS (DESTROY f3))],
[belongs-to f3 f1], [MIL-ACTION f1 f2 r], **rcontrol**
[(during int1)(OCCURS (ACHIEVE (execution-of ma)))],
[(during int)(OCCURS (ACHIEVE (execution-of ma)))]
→ [belongs-to eGforce f1] ;

[P6b] **fn**, [(between int1)(OCCURS (DESTROY f3))],
[belongs-to f3 f1], [MIL-ACTION f1 f2 r], **rcontrol**
[(during int1)(OCCURS (ACHIEVE (execution-of ma)))],
[(during int)(OCCURS (ACHIEVE (execution-of ma)))]
→ [OCCURS (DESTROY eGforce))] ;

If eGforce is now bound to f3, both these sequents would become axioms, as shown below:

[P7a] **fn**, [(between int1)(OCCURS (DESTROY f3))],
[belongs-to f3 f1], [MIL-ACTION f1 f2 r], **rcontrol**
[(during int1)(OCCURS (ACHIEVE (execution-of ma)))],
[(during int)(OCCURS (ACHIEVE (execution-of ma)))]
→ [belongs-to eGforce f1], [belongs-to f3 f1] •

[P7b] **fn**, [(between int1)(OCCURS (DESTROY f3))],
[belongs-to f3 f1], [MIL-ACTION f1 f2 r], **rcontrol**
[(during int1)(OCCURS (ACHIEVE (execution-of ma)))],
[(during int)(OCCURS (ACHIEVE (execution-of ma)))]
→ [OCCURS (DESTROY eGforce))],[OCCURS (DESTROY f3)] •

Notice that '(OCCURS (DESTROY f3))' here matches with '((between int1)(OCCURS (DESTROY f3)))' The expression without time specification refers to the current time in the current world state, and this time is (between int1) because int1 is the time when ma occurs. Note that reducing the above two sequents to axioms does not by itself prove the problem we started with. ***All the sequents in the leaf nodes should be reduced to axioms.*** I have not here shown what happens to sequents [P5c] and [P5d]. Actually, the relationship between **success** and **fn** is (**success** IFF **fn**); i.e., the expression '(**fn** IMPLIES **success**)' is on the left side of all the sequents. This will cause [P5c] and [P5d] to reduce to axioms.

Note that if the boxed AND in **beh** above is changed to OR, then the above proof will not succeed, because in this case, in every military action DESTROY action need not occur against the forces; only either the DESTROY action or **rcontrol** need occur. Thus, [P1] is no longer always true. In this case, the application of [OR →] rule to eliminate the OR connective in **beh** will produce instead of [P7a] and [P7b] above the following four sequents, among which only [P8a] and [P8b] are axioms. The other two will not get reduced to axioms, and thus [P1] will not be proven:

[P8a] **fn**, [(between int1)(OCCURS (DESTROY f3))],
[belongs-to f3 f1], [MIL-ACTION f1 f2 r],
[(during int1)(OCCURS (ACHIEVE (execution-of ma)))],
[(during int)(OCCURS (ACHIEVE (execution-of ma)))]
→ [belongs-to eGforce f1], [belongs-to f3 f1] •

[P8b] **fn**, [(between int1)(OCCURS (DESTROY f3))],
[belongs-to f3 f1], [MIL-ACTION f1 f2 r],
[(during int1)(OCCURS (ACHIEVE (execution-of ma)))],
[(during int)(OCCURS (ACHIEVE (execution-of ma)))]
→ [OCCURS (DESTROY eGforce))],[OCCURS (DESTROY f3)] •

[P8c] **fn**, [MIL-ACTION f1 f2 r], **rcontrol**
→ [belongs-to eGforce f1], [belongs-to f3 f1] ;

[P8d] **fn**, [MIL-ACTION f1 f2 r], **rcontrol**
→ [OCCURS (DESTROY eGforce))],[OCCURS (DESTROY f3)] ;.

This example shows that the property specified by [P1] was true in all world states, for the definition of **beh** given in Fig. 2.1. We started the proof keeping the world state associated with the root empty. One may also, of course, have problems where the properties to be proven are specific to given world states. In these cases, the world state associated with the root will not be empty. My objective in presenting the above example was to illustrate the central aspect of action analysis in CK-LOG: the way *function* and *behavior* of an action are used to predict what might happen in a world state as a result of the action. The expressions that were introduced on the left side in sequent [P4] above are called *action predicates.* Action predicates are used in CK-LOG to reason about the effects of actions on a world state.

Proofs like these are used in CK-LOG not only to prove that certain asserted properties are true, but also to generate plans and to execute actions. The inference rules that are used for analyzing quantified expressions and operator expressions are presented in Appendix B. Many of the rules presented in this Appendix are used in Section 5.

During the proof construction process, CK-LOG will use properties of objects and actions deduced in the proof to update its world states, and the facts available in the world states will in turn be used to control the deductions performed in the proof generation process. In the above example, the only change that occurred in the world state associated with the problem sequent [P1] was the introduction of ma. But if actions are analyzed in greater detail, then the effects of the actions deduced by the theorem-prover will be used to change the world states associated with the sequents.

Every time a world state is to be changed, the inference engine will issue to TMS (the Truth Maintenance System; this is the system used for building models of world states) the appropriate commands to change the world state. When TMS attempts to change the world state, one of three things may happen:

1. The changes are accepted;

2. The changes are conditionally accepted because the world state does not have enough information to check completely whether the changes are acceptable or not;

3. The changes are rejected, because TMS detects a contradiction in the world state as a result of the changes introduced into it; i.e., finds out that for some atom '(r a b),' it is simultaneously true and false.

If they are conditionally accepted, then TMS will return to the inference engine the logical condition, say C, under which the changes would be acceptable. C will be the weakest condition under which the changes will be acceptable. This condition would represent the information that is relevant to the solution of the problem, but is currently unknown in the world state. The inference engine will use this condition either to set up new problems or to modify the existing problems in its deduction tree.

The inference engine thus uses the world state and the knowledge base to introduce into the problems that are being solved, the knowledge that is pertinent to the solution of the problems. CK-LOG has special facilities to mediate communication between its inference engine, its world states, and its knowledge base. These special facilities are presented in the next subsection.

3.3. Communication Between Inference Engine, Knowledge Base, and World States

As mentioned above, the communication between the inference engine, the knowledge base, and the world states is intended to serve two basic purposes:

1. To update problem sequents at the leaf nodes of a deduction tree with knowledge from the knowledge base K[U], and

2. To update the world state based on the findings resulting from the axiom tests performed on the leaf sequents in the deduction tree.

Each problem sequent, Q, in the deduction tree will have a unique world state, U_t, associated with it. It will also have associated with it a set of problems called *hypothesis problems*, denoted by H[Q]. The hypothesis problems that are associated with Q will depend on information that is unknown in the world state but is needed to solve Q. They are generated during the problem solving process as discussed below. The *problem state* defined by Q will consist of the following:*

[Q, (predecessor-of Q), (successor-of Q), (binding-conditions-of Q)
(assignments-of Q), (substitutions-of Q), (world-state-of Q),
(hypothesis-problems-of Q)].

The *context* of a world state, U_t, is the set of sequents defined by:

$$(\text{context-of } U_t) = \{Q \mid (\text{world-state-of } Q\ U_t)\}.$$

Let **F** be the set of sequents in the leaf nodes of the deduction tree. **F** is called the *frontier set* of the deduction tree. At the beginning of the problem solving process, the frontier set, $\mathbf{F}_0 = \{Q_0\}$, where Q_0 is the problem sequent at the root of the deduction tree. The world state, U_0, associated with Q_0 will be the *current world state* that existed at the time this problem was posed.

When the successors of Q_0 are spawned, the frontier set of the deduction tree will change. Let $\{\mathbf{F}_1\}$ be this new frontier set. The context of the world state, U_0, will then change to $[\{Q_0\} \cup \mathbf{F}_1]$. This process of changing the context of the world state will continue as the deduction tree grows. So far, the world state itself has not changed. Changes to the world state may occur when an axiom test is performed on one of the leaf sequents in **F**.† As shown in the examples discussed in the previous section, the performance of axiom tests causes variables in a sequent to be bound to constants. When some variables are thus bound to constants, certain sequents in the frontier set of a deduction tree may reduce to axioms. These sequents that are reduced to axioms will then acquire new atoms, as in

(loves p p),(loves uGx uGx) → (loves p eGy), (loves p p) •

*The reader may note that I am writing this as though the sequent Q itself is an instance of the concept SEQUENT, with *structures* defined for it. Indeed, this is the way the inference engine is implemented in CK-LOG. Notice that (context-of world-state-of) is a converse pair.

†Before performing the axiom test, the sequents Q in **F** will at first be augmented with information obtained from the knowledge base K[U]. The nature of this augmentation will depend on the logical restrictions in K[U], that are associated with the atoms that appear in Q. The augmentation process is described in Ref. 1. We will not be using this process in the examples discussed in this report.

These new atoms are obtained by substituting for the variables their respective bindings; namely, by substituting p for uGx, and p for eGy. Let us call this substitution α, α = [uGx/p, eGy/p]. The above sequent is produced by performing the substitution α on

(loves uGx uGx) → (loves p eGy) ;

This substitution α will be performed on all the leaf sequents in the frontier set of the deduction tree.* This might cause a subset of these sequents to acquire new atoms in this process. Let F_S be the subset of sequents so modified.

The atom '(loves p p)' that was added to the above sequent by the substitution α does not contain any variables in it. Such an atom is called a *grounded atom.* Also, this atom occurred on either side of the sequent obtained after substitution. Such a grounded matched atom is called an *offspring.*† A sequent that is reduced to an axiom may, of course, have several offspring (but usually it will have only one). Each offspring that appears in F_S will be asserted into the world states associated with the respective sequents in F_S. Also, every time a new constant is created in the axiom testing process, this constant will be created in the world state also. If (offspring$_1$, offspring$_2$, ..., offspring$_n$) are all the offspring associated with the same world state, then the inference engine will assert these into the world state by posing to itself the problem:

(ASSERT (offspring$_1$, offspring$_2$, ..., offspring$_n$)) → ;

ASSERT statements that appear on the left side of '→' in a problem are unconditionally asserted into the world state. Assertions like these, if they are successful, will cause the world states associated with the sequents to change. Thus, as the deduction proceeds, different sequents in a deduction tree may have different world states associated with them.

If the assertion is rejected by TMS because of a contradiction that it causes in the world state, then the axiom test is considered to have failed. In this case, either the inference engine will have to backtrack to a previous point in its deduction tree, or try another binding possibility, if one exists. The important point to note here is the following: *the world state determines the success or failure of axiom tests.*

Assertions into a world state may occur not only by axiom tests, as described above, but also through the analysis of operator expressions as discussed in Section 5. The analysis of a CREATE expression may result in asserting into a world state the actions needed to do the creation. Such assertions may appear either on the left or on the right side of '→' in a sequent. The interpretation associated with an ASSERT expression on the right side is different from the interpretation presented above.

> An atom in an ASSERT expression on the right side will be asserted into a world state only if there is a matching atom for it on the left side, separated from other expressions on the left side by commas. A negated atom in an ASSERT expression on the right side will be asserted into a world state only if there is a matching atom (without the negation) for it also on the right side.

The conjunction of atoms in such an ASSERT statement for which matching atoms could not be found will be presented to the user. The user may at that point either reject the assertion, or cause the system to accept the assertion as a hypothesis. If it is asserted as a hypothesis, the system will make a note of it by introducing into the deduction tree the appropriate hypothesis problem.

---

*If the substitution is performed on a sequent which does not contain any of the variables specified in the substitution, then it will leave the sequent unchanged.

†When the mating algorithm mates atoms in a sequent, offspring are produced!

In general, an ASSERT expression may have as arguments arbitrary logical expressions. When the argument of an ASSERT is a complex logical expression, the inference rules for ASSERT operator are used to reduce the logical expression to its component atomic parts, and the resultant atoms are asserted. Examples of assertions of this kind occur in Section 5.

An important property of the inference rules displayed here and in Appendix B is that the rules can be run both ways, from bottom to the top, as well as from the top to the bottom; i.e., starting from a set of axioms, one may apply the rules in the reverse order to conjecture the theorems that give rise to them. (In fact, this is the way Gentzen viewed the system that he proposed.) The significance of this in a military planning environment is that one may use CK-LOG to conjecture about enemy objectives starting from the actions and situations that exist in a world state. This problem needs further study. The concept of a plan in CK-LOG is presented in the next section.

3.4. CK-LOG's Concept of a Plan

Plans are extracted in CK-LOG from proof trees. Thus, the plan for realizing a military objective will be extracted from the proof tree associated with the achievement of this objective. As mentioned earlier, each node in a proof tree is a problem to be solved, stated as a sequent. Thus, a proof tree associated with the realization of a military objective will represent the set of problems that should be solved in order to achieve the objective, and the way their solution is achieved. The entire proof tree will thus represent the detailed plan for the achievement of the objective. The variable bindings used in the proof tree will represent the objects, places, times, forces, and actions that are used to achieve the objective, and the order in which they are used, with associated timing and resource restrictions, if any.

Usually there is no need to state a plan at this level of detail. Eliminating some of the details in a proof tree serves two purposes: this makes the plan statement brief, and also allows for possible variations in a plan at the time of its execution. The way a problem was solved at the time of plan generation may not exactly coincide with the way it gets solved at the time of plan execution. By representing in a plan only the important and critical problems extracted from a proof tree, and dropping off the details of problem solution, one allows for the possibility that these problems might get solved differently at the time of plan execution. The problem details and variable bindings presented in a plan would represent the details that one ought to expect at the time of its execution. During execution if one or more of the problems stated in a plan does not occur, or occurs in a different form, then this will indicate a departure from the plan, calling for either plan revision or action modification. Thus, military plans represented in CK-LOG may be used to guide the plan execution:

- The problems in the plan may be used to prompt a commander on needed actions, their timing and resource requirements, and check the changes introduced in a world state by the actions.
- The problems and the bindings in the plan may be used to monitor plan execution and recognize departures from the plan during execution.
- The system may be used to analyze the new problems encountered at the time of plan execution and perform the necessary plan revisions.

The plan (proof) generation process is partly illustrated in Section 5. I do not discuss in this section the plan extraction process itself, but point out how the proof contains the information needed to develop an OPPLAN. Let me now conclude this section with a summary of the architecture of CK-LOG.

3.5. The Meta Description System and The Architecture of CK-LOG

The architecture of CK-LOG is shown schematically by the block diagram in Fig. 3.3. The two principal components of the system are represented in this diagram by the two L-shaped regions with bold outlines. The outer L-region is MDS consisting of its three components, the *knowledge representation system* (KRS), the *knowledge base* (KB), and the *truth maintenance system* (TMS) associated with the *world states*. The inner L-region represents the *theorem-proving system* (TPS) consisting of its three components, *meta-system for defining inference rules* (MSI), the knowledge base of *inference rules* (KBI), and the *inference engine* (IE) with its associated *problem states*. MSI is implemented using KRS. The rules in KBI are represented using structures similar to those in KB. IE is partly implemented using the facilities available in TMS, and partly through specialized routines written in ELISP. The block between the two L's, called UI, is the *user interface*.

The commands to the truth maintenance system may either update the world state, i.e., create or destroy objects and actions in the world state, or define or modify their properties, or check the truth of logical expressions in the world state. The basic commands are: CRI for create instance, DESI for destroy instance, ASSERT for adding properties of objects and actions in the world state, and CKECK for checking the truth values of logical expressions in a world state.

When an ASSERT or a CRI command is issued to the truth maintenance system, it will first produce a new world state incorporating into it all the changes specified by the command. It will then check whether this new world state satisfies all the restrictions associated with the dimensions in **K[U]** that are relevant to the changes just made. As mentioned earlier, this will result in one of three possible outcomes: either it is accepted, conditionally accepted, or rejected.

An important requirement in the definition of **K[U]** is that the consistency conditions should not contain any operator expressions. This will make sure that the reasons returned by the truth maintenance system would not contain operator expressions. This is a necessary condition for the formulation and use of hypothesis problems in the proof procedure.

The constants created during an axiom test are introduced into the world state by using either one of the commands:

(CRI c **range**) → ;
(ASSERT (*instance-of* c **range**)) → ;

to the inference engine, where **range** specifies the range restriction on the constant, c. A CRI command should never appear on the right side of a sequent, for it has no interpretation on the right side.

The Meta Description System, MDS, is the basic building block of CK-LOG. It uses **K[U]** to model the world states of the universe **U**. MDS is used in CK-LOG to implement its problem solving system. As shown in Fig. 3.3, the problem solving system of CK-LOG also has three components: the *meta system for defining inference rules*, the *knowledge base of inference rules*, and the *inference engine & problem state*. These correspond to the three basic systems of MDS. The knowledge representation system of MDS is used to define the concepts, SEQUENT, PROBLEM, AXIOM, INFERENCE-RULE, DEDUCTION, PROBLEM-STATE, etc., that are needed to implement the problem solving system of CK-LOG, and the truth maintenance system of MDS is used to model the problem states of the inference engine. Just as models of world states contain constants which are objects and actions in the universe, **U**, models of problem states will contain constants which are sequents, deduction trees, and actions associated with the application of inference rules and axiom tests. These actions are, however, implemented as LISP functions. Thus, the blocks inside the inner L of the block diagram shown in Fig. 3.3 are implemented using mostly the facilities of MDS itself.*

*The implementation of the inference engine has not been completed yet.

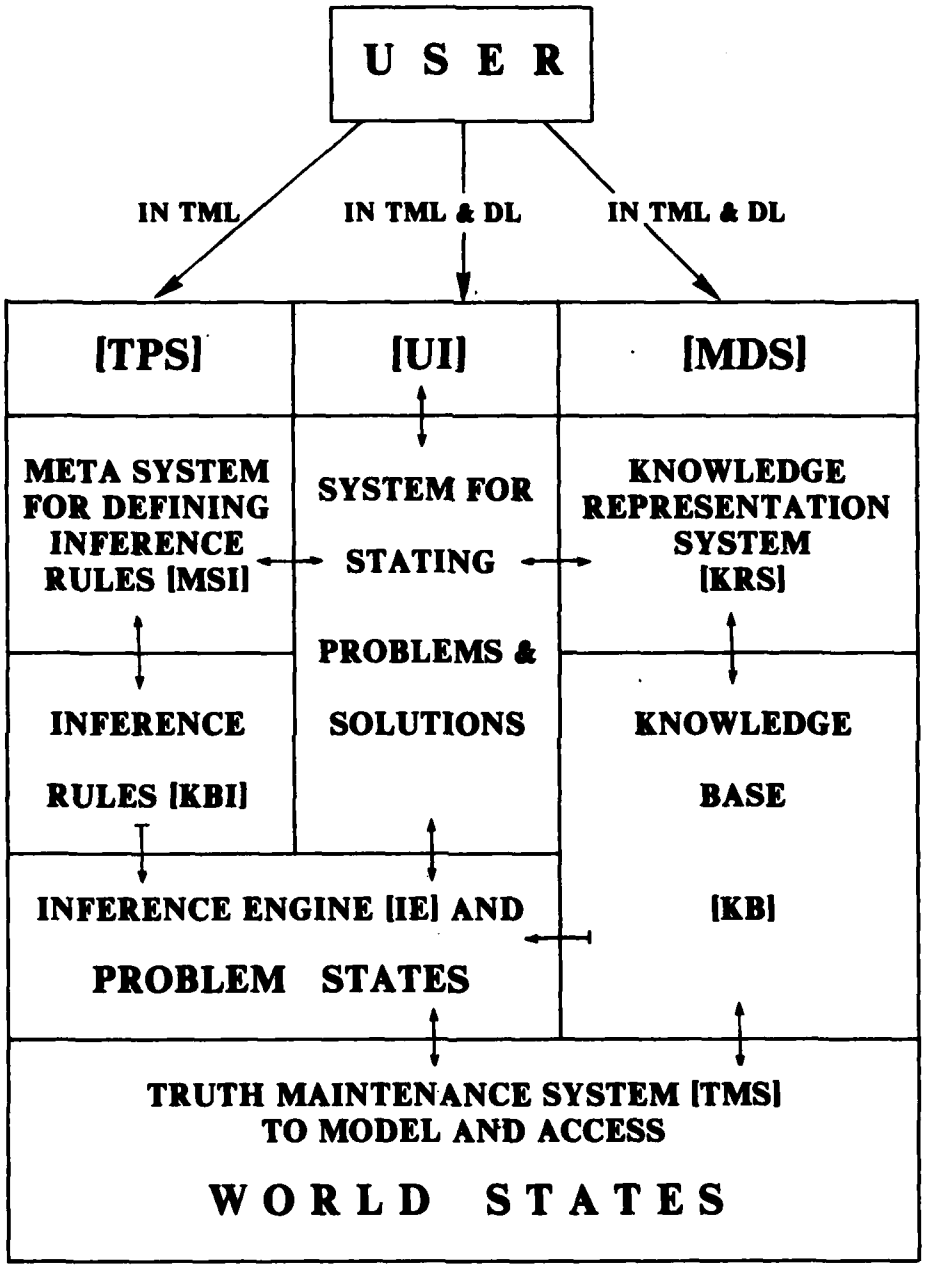


Fig. 3.3—The architecture of CK-LOG

With this introduction to CK-LOG, let me now proceed to the discussion of the problem solving adequacy of CK-LOG for Naval Operational Planning.

4. THE PROBLEM SOLVING ADEQUACY: OPPLAN DESIGN DESCRIPTION

The main focus in this section and the next is on two basic issues: how may a user present his problems to CK-LOG, and how will CK-LOG respond? In the discussions here, three categories of information are presented: the first is general information on the operational planning guidelines given in NWP-11; the second is the specific set of problem solving processes that are triggered in CK-LOG by OPPLAN *design* statement; the third is the manner in which the NWP-11 guidelines are captured by these problem solving processes. These three categories of information do not, however, always appear in separate sections; they are intermixed. I chose this mode of presentation to maintain emphasis and interest on the main focus.

The method for plan development is given by OPPLAN *design*. To understand this *design* specification, there are three pieces of information that one should know: (a) The OPPLAN-CONSULTANT user context in which the *design* is invoked; (b) The planning process in NWP-11 which it attempts to capture; and (c) The inference rules of CK-LOG that make it possible for CK-LOG to use the *design* description and develop the plan. These are discussed in this section.

A discussion of how the OPPLAN *design* statement, the knowledge base, the world states, and the inference rules are used by CK-LOG to set the planning problems and solve them appears later in Section 5. Together, Sections 4 and 5 illustrate the problem solving adequacy of CK-LOG for Naval Operational Planning.

4.1. The Planning Process and the User Context

Assume that a COMMANDER has already received an OPORDER, and that he has begun the design of an OPPLAN for the OPORDER-TASK which has been assigned to him. In the example given in NWP-11, this commander is RadmWB (Rear Admiral WB, Commander of North Pacific Attack Carrier Striking group). The oporder received by him was issued by AdmWA (Commander of White North Pacific Forces, CTF-19). Using the terminology of NWP-11, we will say that this oporder defines the mission of the RadmWB. Let oporder-task be the OPORDER-TASK in the oporder that is assigned to RadmWB.

The geographical region of our example is the north Pacific region. The name Northern-Pacific is used below to refer to this region. Assume that the various regions in the Northern-Pacific, including the northern Asiatic and American regions, the various islands, peninsulas, oceans, seas, and cities have all been already specified in terms of suitably defined BASIC-REGIONS. Also, that the descriptions of regions specify the forces that control them, can defend them, and can attack them, the installations that they contain, and the forces they contain. The two contending forces are White and Green. Assume that the oporder, oporder-task, RadmWB, and AdmWA are in this world state model. The oporder of our example is received by the commander of the White forces in the Northern-Pacific region.

At this point, the first task for a user will be to create a new instance of OPPLAN to represent the plan that is going to be created. The next task is to bring the world state up to date with the military situation description given in the oporder. After this is done, the user may begin the planning process in CK-LOG by attempting to ACHIEVE the objective-of the newly created OPPLAN. The OPPLAN *design* specification in Fig. 4.1 shows the formal statement of the last two tasks mentioned above. As shown in Section 5, this brief statement will cause CK-LOG to spawn a whole spectrum of planning and problem solving activities that are appropriate to the given mission of RadmWB.

Let opplan be the new instance of OPPLAN created by the user. The initialized properties of this opplan will be the (oporder-for opplan), (oporder-task-for opplan), (created-by opplan), (objective-of opplan),

(purpose-of opplan), and (is-mil-plan opplan). The user will create and initialize this opplan in the world state of CK-LOG by issuing the command:

(CRI opplan OPPLAN
(oporder-for opplan oporder) (oporder-task-for opplan oporder-task)
(created-by opplan) (objective-of opplan)
(purpose-of opplan) (is-mil-plan opplan)),

where CRI stands for 'CReate Instance.' This function will create a new instance of OPPLAN, called opplan, in the existing world state and assert the specified properties. Notice that the assertions give values only for the first two properties; namely, for oporder-for and oporder-task-for. For the remaining properties, to find the appropriate values, CK-LOG will use the relevant restrictions from the OPP-RESNs specified in Section 2.3.1. Thus, by using OPP-RESN1, it will set (created-by opplan) = RadmWB, and by using OPP-RESN10 it will set (is-mil-plan opplan) = mil-action that is specified in the oporder-task; similarly, the objective and purpose of the opplan will also be set to be the same as those of the oporder-task. The CRI command will succeed only if all the specified properties are successfully instantiated (i.e., assigned proper values).

In fact, in CK-LOG, relations like these are specially marked in the definition of a concept indicating to the system that they have to be defined at the time of creation of a new instance of the concept. If the user did not specify them and if the values were not automatically determinable from the restrictions associated with the properties, then the system will prompt the user for their values. Relations like these are called *prompt relations* in CK-LOG.

4.1.1. The Results of the Planning Process

The initial specification of the mil-action in opplan, will be incomplete. For example, it may not specify all the assigned forces or give the specific objectives, the details of force deployment, engagement, needed logistics support details, and communication policies. The planning process should result in the completion of these missing details. This may take one of two forms: in the first and simple case, the completion is achieved directly by correctly identifying all the missing properties of the given mil-action and filling them in. In this case, opplan will specify only one opplan-task. The military action of this opplan-task will be this mil-action itself. The objective of this opplan-task will be the same as that of the (oporder-task-for opplan), and the assigned commander of this opplan-task will be RadmWB himself. Here, the planning process will not result in the creation of any new operational orders. This will represent a terminal plan in the plan hierarchy.

In the second and more complicated case, planning will result in the specification of several OPPLAN-TASKs which together fulfill the requirements of the commander's mission. The objectives of these OPPLAN-TASKs would have been derived from the objective of the opplan as a part of the planning process. In this case, the resulting opplan will give rise to a new operational order, which in turn will cause new planning processes to be initiated by the commanders who received it. The example discussed in this chapter pertains to this second case. Let us now get back to opplan design.

Having created the new instance, opplan, to initiate the opplan design process, the user will tell the system to (ACHIEVE (*design* opplan)). Generally, if the *design* specification exists for the concept under consideration (in the above case the concept is OPPLAN), then the system will use it to set up for itself the appropriate subgoals. If it does not exist, then it will follow the default procedure for completing the description of the given instance of the concept. This default procedure will merely consist of attempts to create values for the various properties of the instance as it is defined in the concept definition. In all cases, where the design is complicated, a *design* description will invariably exist for the concept. The nature of this *design* specification for OPPLAN, and the way it relates to the planning process described in NWP-11, are discussed in the next subsection.

4.1.2. The Planning Process in NWP-11

The design of an OPPLAN is viewed in NWP-11 in terms of two clearly identified stages: the first stage is called *development of the commander's estimate of the situation.* The information generated during this stage of the planning process will be summarized by the additional OPPLAN structures shown in Fig. 4.1. This stage will end with the so-called *commander's decision*, which is represented in Fig. 4.1 by the COMNDR-DECISION concept.

Structure: (cand-phy-objectives-for OPPLAN OBJECTS), 1
(cand-own-courses-of-action-for OPPLAN MIL-ACTIONS), 1
(cand-own-courses-of-action-for OPPLAN MIL-OBJECTIVE), 1
(opposing-courses-of-action-for OPPLAN MIL-ACTIONS), 1
(opposing-courses-of-action-for OPPLAN MIL-OBJECTIVE), 1
(action-comparisons-for OPPLAN ACTION-COMPARISONS), 1
(commanders-decision-for OPPLAN COMNDR-DECISION), 1

Design: [(after (*time-of* (ASSERT (situation-of (oporder-for opplan)))))
(ACHIEVE (objective-of opplan))]

COMNDR-DECISION:
(is-commanders-decision COMNDR-DECISION OPPLAN), 1
(selected-course-of-action-of COMNDR-DECISION MIL-ACTION), 1
(justification-of COMNDR-DECISION ACTION-COMPARISON), 1

Fig. 4.1—Design specification for OPPLAN

The second stage is called the *development of the plan and directive.* The second stage will, of course, depend on the findings obtained from the first stage of the process. This will end with the plan specification and distribution of new oporders, if there are any. There is also a stage of the planning process where a commander may develop a plan for the planning process itself. This will be necessary only for the development of complex plans with given time deadlines. The discussions in this report pertain only to the first stage of the planning process; namely, the development of commander's estimate of the situation. This is sufficient to illustrate the problem solving adequacy of the system. Let me briefly elaborate on this stage of planning.

4.1.2.1. Commander's Estimate of the Situation and the OPPLAN Design Specification

The development of commander's estimate is subdivided in NWP-11 into five parts:

- Mission and its analysis,
- Considerations affecting possible courses of action,
- Analysis of opposing courses of action,
- Comparison of own courses of action, and
- Commander's decision.

The structure definitions shown in Fig. 4.1 are intended to represent the results of the above five-part analysis: it should result in the enumeration of all the candidate physical objectives (cand-phy-objectives-for), the candidate own courses of action for (cand-own-courses-of-action-for) the opplan, and the possible opposing courses of actions. Once these are obtained, the commander may choose one or more of the physical objectives as candidates for further exploration and compare available own courses of action against the possible enemy courses of action. The results of this comparison are represented by the ACTION-COMPARISON concept. This comparison will be used to make an informed choice of own course of action from the list of (cand-own-courses-of-action-for opplan). The particular choice that is made by the commander is called the commander's decision, represented by (selected-course-of-action COMNDR-DECISION) and its justification. The justification is obtained from the action comparisons.

The problem solving process that is used to generate this information will in fact make precise what this five-part analysis is and how it is done. Note that NWP-11 itself gives only general guidelines for this five-part analysis process; it does not give the problem solving methods that a commander might use. We see in Section 5 a specific manifestation of the problem solving methods that this analysis entails.

In Fig. 4.1 the relations cand-own-courses-of-action-for and opposing-courses-of-action-for may have values that are either MIL-OBJECTIVEs, or MIL-ACTIONs. In complex plans, the initial analysis might not directly result in the identification of the available military actions. Instead, it may result in the transformation of the given objective to combinations of other simpler objectives, which together may lead to the realization of the mission. This is called the *problem reduction* process. The MIL-ACTIONs to realize these objectives will eventually result from the analysis of these transformed objectives. An example of this process occurs in the solution of the problem is introduced in the next section.

4.1.3. A Planning Example

4.1.3.1. The Mission Purpose and Objective

Suppose that the mission of RadmWB is specified by the following*:

(objective-of oporder-task) = (objective-of opplan) = RadmWB-objective:
[(during [(1984 June) (1984 Sept)])
(PREVENT
((EXISTS base (bases-at Attu-Island))
([belongs-to base Green] AND
[type-of base advanced] AND
[status-of base operational]))]

Assume here that in the current world state it is known that Green is the enemy force, and White is the force to which RadmWB and his assigned forces belong. For the purpose of discussion here, assume also that the type-of a BASE is advanced if one can mount offensive operations from the base. The status of a BASE is operational if the base can be used to perform its *function*. The statements of the purpose and objective of the oporder received by RadmWB, taken from NWP-11, Appendix F, are the following:

AdmWA-Objective: To take control of Attu-Island away from Green after Sept. 1984.

AdmWA-Purpose: To avert Green expansion in the North-Pacific area in order to assist in the protection of Alaska and the continental United-States from sea or air attack.

* This example is taken from Appendix F of NWP-11.

RadmWB-Objective: Prevent the establishment of an advanced operational base at Attu-Island by Green.

RadmWB-Purpose: = AdmWA-Objective.

Notice that RadmWB-Purpose is the same as AdmWA-Objective; i.e., this will be the purpose statement in the oporder received by RadmWB. The formal statements of AdmWA-Objective and AdmWA-Purpose in the language DL are shown in Fig. 4.2. The statement of AdmWA-Objective is quite straightforward, but that of AdmWA-Purpose is rather complicated and needs some explanation. As we know, forces have the offensive-range-of property defined for it (see Fig. A6 in Appendix A). $Region_1$ is within the expansion-of the offensive-range-of Green and is in the North-Pacific (the expansion-of a region x is a region y such that x is within y). Also, $Region_1$ is such that it is not in the offensive-range-of Green in the current world state.

(objective-of oporder) = AdmWA-objective:
[(after [1984 Sept 1]) (CREATE (controls Attu-island White))]

(purpose-of oporder) = AdmWA-purpose:
[(EVERY $region_1$ (expansion-of (offensive-range-of Green)))
([([is-within North-Pacific $region_1$] AND
[NOT (is-within (offensive-range-of Green) $region_1$)]) IMPLIES
(PREVENT (CREATE (offensive-range-of Green $region_1$)))] AND
[SUPPORT
(DESTROY
((EXISTS $region_2$ (is-within Alaska), (is-within Continental-US))
([offensive-range-of Green $region_2$] AND
[(EXISTS mil-action NAVAL-MIL-ACTION, AIR-MIL-ACTION)
([prosecuted-by mil-action Green] AND
[region-of mil-action $region_2$])])))])]

Fig. 4.2—The objective and purpose of OPORDER

The following should now be prevented: that $region_1$ is brought within the offensive-range-of Green. This is expressed in the statement of Fig. 4.2 by the '(PREVENT (CREATE --))' expression. This prevention should be such that it supports the destruction of the truth of the following:

> That there is a region, $region_2$, that is in Alaska or in Continental-US, such that it is in the offensive-range-of Green, and is the region-of an air or naval mil-action prosecuted by Green.

This is expressed by the '(SUPPORT (DESTROY --))' expression in Fig. 4.2. This captures the concept, 'in order to assist in the protection of Alaska and Continental-US ...;' It might not by itself achieve the objective of eliminating all naval and air attacks on Alaska and Continental-US, but it should support it. Here NAVAL-MIL-ACTION and AIR-MIL-ACTION are being viewed as concepts that are specializations of MIL-ACTION. Let me now proceed to the description of the military situation of our example.

4.1.3.2. The Military Situation

The following abstracts taken from NWP-11, Appendix F, present the relevant picture of the military situation at the time the oporder was given.

> Green and White have been at war for eighteen months. During the first month of the war, White's Aleutian-Island and Alaskan bases were destroyed by Green. Kodiac-Island, with the only remaining major White naval base in the Alaskan area, was left undamaged. ... Green has occupied Attu-Island without opposition. ... White is preparing to recapture Attu-Island by amphibious assault on 1 Sept. 1984. Its accomplishment requires that Green be prevented from establishing effective air or naval bases on Attu-Island.

Normally, the statement of a military situation may specify five kinds of information: enemy-forces-in the situation, friendly-forces-in the situation, enemy-objectives if known, objectives of the friendly forces in the situation, and military actions that are occurring in the situation. None of these might be specified in full detail. The above description specifies that some military actions had been going on for the past 18 months, but does not give details of this action, except for the fact that all the White's bases in Alaska and Aleutian-Islands have been destroyed by Green. It also states a friendly-objective that is relevant to the oporder. The formal statement of this in DL is shown below in Fig. 4.3. I have assumed that situation-desn is the description of the (*situation-of* oporder), where oporder is the operational order received by our RadmWB, and mil-situation is the military situation that situation-desn refers to. It is true at the (*time-of* (has-received RadmWB oporder)), which has been assumed to be (1984 May 25).

The formal description shown in Fig. 4.3 is a conjunction of eight conjuncts. The first three simply specify the friendly objectives in the military situation, the ones we have already seen. The remaining five are paraphrased below:

Conjunct-4: Military actions prosecuted by White or Green had been occurring between (1982 Dec 1) and (1984 May 25) and these are the mil-actions-in the mil-situation that is being described.

Conjunct-5: All the White bases in Alaska and Aleutian-Islands have been destroyed by Green between Dec. 82 and Jan. 83.

Conjunct-6: White's base at Kodiac-Island is still operational.

Conjunct-7: Green has control of Attu-Island.

Conjunct-8: There is a plan, opplan_0, for an amphibious assault, amphi-assault, that is currently approved and will become active in Sept. 1984. The objective of this plan is AdmWA-objective, and RadmWB-objective is the objective of an opplan-task of this plan.

Since RadmWB-objective is the objective of an opplan-task of the plan for amphi-assault, namely, the plan, opplan_0, it follows that RadmWB-objective is required for the success of the amphi-assault. Clearly, it is this opplan_0 that should have given rise to the oporder that RadmWB received. The purpose of this plan is the AdmWA-purpose mentioned earlier and its objective is the AdmWA-objective.

4.1.4. Setting Up the Planning Problem

This then is the (situation-of (oporder-for opplan)) when the user gives the command:

→ (ACHIEVE (*design* opplan)) ;

In response to this command, CK-LOG will set up its first goal sequent, seq-1. Let *goal-1* = (*design* OPPLAN); namely,

Goal-1: [(after (*time-of* (ASSERT (situation-of (oporder-for opplan)))))
(ACHIEVE (objective-of opplan))],

seq-1: → *goal-1* ;

(situation-of oporder) = situation-desn.
Mil-situation is being described by situation-desn below.
(*time-of* (has-received oporder RadmWB)) = (1984 May 25).

situation-desn:

[(EXISTS mil-situation MIL-SITUATION)
([friendly-objectives-in mil-situation AdmWA-objective]
AND
[friendly-objectives-in mil-situation RadmWB-objective])
AND
[friendly-objectives-in mil-situation AdmWA-purpose]
AND
[(EXISTS mil-action (mil-actions-in mil-situation))
([(between ((1982 Dec 1)(1984 May 25))) (OCCURS mil-action)] AND
[(prosecuted-by mil-action Green) OR (prosecuted-by mil-action White)])]
AND
[(EVERY base (bases-of White)]
([NOT (isin base Kodiac-Island)] AND [(isin base Aleutian-Island) OR (isin base Alaska)])
IMPLIES [(between [(1982 Dec) (1983 Jan)])(destroyed-by base Green)]))
AND
[(EVERY $base_1$ BASE)
([isin $base_1$ Kodiac-Island] AND [status-of $base_1$ operational] AND [belongs-to $base_1$ White])]
AND
[controls Attu-Island Green]
AND
[(EXISTS amphi-assault AMPHIBIOUS-ASSAULT)
([prosecuted-by amphi-assault White] AND [region-of amphi-assault Attu-Island] AND
[is-against amphi-assault Green] AND
[(EXISTS $opplan_0$ (mil-plan-for amphi-assault))
([gives-rise-to $opplan_0$ oporder) AND [created-by $opplan_0$ AdmWA] AND
[status-of $opplan_0$ approved] AND [objective-of $opplan_0$ AdmWA-objective] AND
[CREATE ((*at* (1984 Sept 1)) (status-of $opplan_0$ active)]) AND
[(EXISTS opplan-task (specifies-tasks $opplan_0$))
([objective-of opplan-task RadmWB-objective] AND
[assigned-comndr-of opplan-task Radm-WB])])])])]

Fig. 4.3—Military situation of OPORDER

The process of achieving this goal is the planning process in CK-LOG. Note, however, that this goal cannot be deduced from the knowledge base and the information available in the world state, because these will be incomplete. Thus, during the planning process, it will be necessary for the user to provide the missing information, and make the hypotheses when needed. The problem solving process should be able to prompt the user for the missing information and check the information given by the user for consistency. It should also be able to identify the hypotheses on which a plan might be based, or incorporate into its planning process the postulates given by the user. To discuss how this is done in CK-LOG, it is first necessary to introduce the specialized inference rules used by CK-LOG to analyze and satisfy goals like the one above. These are presented in Appendix B. Let me now proceed to the discussion of the planning process.

5. THE PROBLEM SOLVING ADEQUACY: THE PLANNING PROCESS

5.1. Understanding the Mission Objective

This section shows how CK-LOG reduces the construction goal specified in seq-1 of Section 4.1.4 to the relevant set of problems that the user should seek to solve, and in this process displays its ability to "understand" a given mission statement. I show here how CK-LOG first assimilates the current military situation specified in the situation-desn of Fig. 4.3, how it helps the user to correctly assess the scope of operations that he should plan to undertake, and how it provides the user with significant planning alternatives, which he should seek to analyze further. The analysis of these alternatives is discussed in the next subsection.

The table below shows the initial sequence of deductions made by the system starting from the command, [ACHIEVE (*design* opplan)], given by the user. The last column indicates the rules used on the sequent in its row, to get the sequent in the next row below. Most of the rules cited in this column are presented either in Section 3 or in Appendix B. If there are several expressions in the sequent, then the expression to which the rules are applied is marked with a '[#]' at its end. Note that for each sequent, the candidate rules that are applicable to the sequent will be presented to the user by the system. The particular rules applied to a sequent at a given time is selected by the user from the list of candidate rules. In most cases, however, once the user chooses the expression in the sequent on which he wishes to focus, the rule applicable to the expression will usually be unique.

For convenience, I have reproduced below the *goal-1* stated previously and defined the expressions, *event* and *exp*.

Goal-1 = [(after (*time-of* (ASSERT (situation-of (oporder-for opplan)))))
(ACHIEVE (objective-of opplan))],
= ((after (*time-of event*)) *exp*), where
Event = (ASSERT (situation-of (oporder-for opplan))), and
Exp = (ACHIEVE (objective-of opplan)).

Assume that the *event* is not true in the world state. Then, the sequence of deductions shown in Table 1 will occur (note that the rule [→ E] used in the table below is presented in Appendix B, and [→ IMP] was discussed in Section 3). For convenience, I have used above the names situation-desn and RadmWB-objective, instead of the actual expressions that are denoted by these names. If the situation-desn is true in the world state, then one would get seq-6′ directly from seq-6:

seq-6′: situation-desn → (CREATE ((after ..) ...)) ;
(ei-t = (**value-of** (*time-of* (ASSERT situation-desn))))

Table 1—Initial Sequence of Deductions from '*goal-1* →';

Name	Sequents and Bindings	Rules Used
seq-1	→ [ACHIEVE (*design* opplan)] ;	[ACH-EL]
seq-2	→ [CREATE *goal-1*] ;	[EVTM-EL4]
seq-3	→ ([ei-t *event*] IMPLIES [CREATE ((after ei-t) *exp*)])] binding: ei-t = (**value-of** (*time-of event*)).	[→ IMP]
seq-4	(ei-t (ASSERT (situation-of ..))) → (CREATE ((after ei-t) *exp*))) ;	[ASRT-TM] [CRR-OP2]
seq-5	(ASSERT (situation-of ..)) → (CREATE ((after ei-t) (ACHIEVE ..))) ; binding: ei-t1 = ei-t.	[ASRT-TRM] [CRR-OP2] [ACH-EL], [CRR-TM]
seq-6	(ASSERT situation-desn) → (CREATE ((after ei-t) RadmWB-objective)) ;	[ASRT-E →]

where the value of (*time-of* ...) will simply be the time at which the ASSERT was tested as being true. In this case, there would be no need to assert the situation-desn into the world state, and the system would directly proceed to the achievement of RadmWB-objective goal. We have assumed that situation-desn (shown in Fig. 4.3) is not true in the current world state. Thus, seq-6 in Table 1 now represents the two tasks that remain to be performed. The time ordering of the tasks in seq-4 puts ASSERT as the first task. This is therefore taken up as the first goal. Now consider the way this ASSERT statement is analyzed and assimilated.

5.1.1. Assimilation of the Given Military Situation

The analysis of situation-desn is shown in Table 2. In Table 2, the integers appearing within parenthesis in the last column indicate the number of times the rule is applied. Thus, the (3,1,1) label in the row of seq-7 indicates that the [ASRT-E →] rule was applied 5 times in all. It was applied thrice first, then once, and then again once. For convenience, the following abbreviation has been used:

situation-desn = ((EXISTS mil-situation MIL-SITUATION) conjunction),

where conjunction is the conjunction shown in Fig. 4.3. The application of [ASRT-E →] and [ASRT-U →] rules create the variables ei-c1 through ei-c6 and ug-x, and causes these variables to be substituted for their associated quantified variables in the conjunction. After this, the [ASRT-DNF] and [ASRT-AND →] rules will be applied. To simplify the presentation, the application of [ASRT-DNF] rule has been suppressed in Table 2, and only the [ASRT-AND →] rule has been applied. The resultant expression, shown in seq-8 of Table 2, thus still contains within it a disjunction and an implication. The partial ordering on the variables generated by the deductions in Table 2 is shown in Fig. 5.1. At this point, CK-LOG will initiate the actions for assimilating this ASSERT statement into the world state: To do this, CK-LOG will first check the truth of the conjuncts in seq-8 in its current world state (which in this case is the world state associated with the sequent) by assigning values to ei-c's and ug-x from the objects in the world state that satisfy the respective binding conditions. The variables will be chosen for consideration in the order specified in the partial ordering shown in Fig. 5.1.

Table 2—Analysis of Situation-desn

Name	Sequents and Bindings	Rules Used
seq-6	[ASSERT ((EXISTS ..) conjunction)] → ..; binding: ei-t1 = ei-t.	[ASRT-E →](1)
seq-7	[ASSERT ((ei-c1 in MIL-SITUATION), (**subst** ei-c mil-situation conjunction)] → ..;	[ASRT-E →] (3,1,1) [ASRT-U →][ASRT-AND]
seq-8	[ASSERT ([ei-c1 in MIL-SITUATION], [friendly-objectives-in ei-c1 AdmWA-objective], [friendly-objectives-in ei-c1 RadmWB-objective] [friendly-objectives-in ei-c1 AdmWA-purpose], [ei-c2 in (mil-actions-in ei-c1], [((between ((1982 Dec 1)(1984 May 25))) (OCCURS ei-c2)] AND ([prosecuted-by ei-c2 Green] OR prosecuted-by ei-c2 White])], [(ug-x in (bases-of White)) IMPLIES ([(NOT (isin ug-x Kodiac-Island)) AND ([isin ug-x Aleutian-Island] OR [isin ug-x Alaska])] IMPLIES [(between [(1982 Dec) (1983 Jan)]) (destroyed-by ug-x Green)])], [ei-c3 in BASE], [isin ei-c3 Kodiac-Island], [status-of ei-c3 operational], [belongs-to ei-c3 White])], [controls Attu-Island Green], [ei-c4 in AMPHIBIOUS-ASSAULT] [prosecuted-by ei-c4 White], [region-of ei-c4 Attu-Island], [is-against ei-c4 Green], [ei-c5 in (mil-plan-for ei-c4)], [gives-rise-to ei-c5 oporder], [created-by ei-c5 AdmWA], [status-of ei-c5 approved], [objective-of ei-c5 AdmWA-objective], [CREATE ((*at* (1984 Sept 1))(status-of ei-c5 active)], [ei-c6 in (specifies-tasks ei-c5)], [objective-of ei-c6 RadmWB-objective], [assigned-comndr-of ei-c6 Radm-WB])] → ..;	--

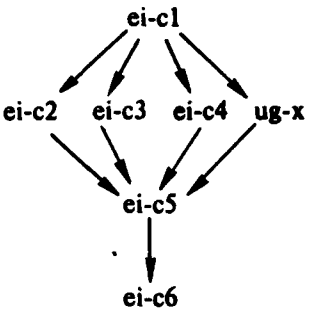


Fig. 5.1—Partial ordering of variables in Table 2

We have already assumed that the military situation, ei-c1, does not exist in the world state. Thus, a new instance of MIL-SITUATION will be created and assigned to ei-c1. This will satisfy the binding condition, '(ei-c1 in MIL-SITUATION).' The specified properties of ei-c1 will also be asserted into the world state model. This will cause the following action to be performed:

(CRI mil-sitn MIL-SITUATION
(friendly-objectives-in mil-sitn AdmWA-Objective)
(friendly-objectives-in mil-sitn RadmWB-Objective)
(friendly-objectives-in mil-sitn AdmWA-Purpose))
(ei-c1 = mil-sitn),

and the results announced to the user. Assume that the objectives and purpose mentioned above are already in the world state. The 'friendly-objectives-in' relation has no restrictions associated with it, and also none of the other restrictions defined for other relations in the knowledge base depend on it. Thus, the above CRI operation will succeed.

Let us suppose that CK-LOG chose next the variable ei-c2 for consideration. As per the binding condition, ei-c2, should be in (mil-actions-in mil-sitn). But no values have as yet been assigned for this relation (it is as yet undefined in the world state). Since (mil-actions-in MIL-SITUATION) = MIL-ACTION, CK-LOG will now look for candidate MIL-ACTIONs in the current world state, assign them one by one to ei-c2, and check whether any of them satisfies the conditions specified for ei-c2 in the ASSERT statement. This is a general strategy followed in CK-LOG: whenever the value for a term is unknown, CK-LOG will search its world state for candidates. In this case, each MIL-ACTION action, that is found to satisfy the conditions (in this case, the conditions are the relations in the ASSERT in which the variable ei-c2 appears) will be assigned as a value for (mil-actions-in mil-sitn).

If no such military action exists, then two new ones will be created (using CRI again) and assigned as values for (mil-actions-in mil-sitn). Two are created here because of the following: notice that the conjunct in seq-8 in which ei-c2 appears has a disjunction in it. When the expression is put in disjunctive normal form, one will get in effect two ASSERT sequents from it (because of the use of [ASRT-DNF], [ASRT-V →], and [V →] rules):

[ASSERT (...([(between ..)(OCCURS ei-c2)] AND [prosecuted-by ei-c2 Green])....)], ... → ;
[ASSERT (...([(between ..)(OCCURS ei-c2)] AND [prosecuted-by ei-c2 White])....)], ... → ;

Since a military action can be prosecuted only by one FORCE (this information is in the *structure* specification of MIL-ACTION), the same binding for ei-c2 cannot satisfy both of these assertions. When CK-LOG attempts to assign two values for the prosecuted-by relation, it will encounter a contradiction in its

world state. However, (mil-actions-in mil-sitn) can have more than one military action in it. Thus, two will be created to represent the two cases above.

We have here a special case. All the variables in the sequent were generated from within the same ASSERT statement, and the dependencies in the model space did not forbid the creation of another instance for ei-c2 in the same world state. In general, two kinds of restrictions may limit the bindings of variables like these. One is the restriction imposed by the order of the variables in the sequents: depending upon the other literals that appear in the ASSERT, this may force the variable to assume the same value in both sequents. Thus, for example, suppose there was a relation that forced an ei-x to be the father of ei-y, and ei-y had been previously bound in both sequents. In this case, ei-x, also should have the same binding in all the disjuncts, and to represent all the disjuncts, one may need different bindings for ei-y itself in the different disjuncts. Other restrictions that control the instantiation of new constants and relations may arise from the world state itself. Thus, for example, if there was a disjunction that said John was either in Washington, D.C. or in Los Angeles, then these two possibilities could not be both represented in the same world state.

When this happens, it is generally indicative of the fact that the disjunctive possibilities cannot all occur simultaneously in the same world state. In cases like this, the different disjuncts will have to be represented by different possible world states. This is done in CK-LOG by associating contexts with world states. The different disjuncts will be represented by world states in different contexts. The sequents themselves will mark these different contexts.

It is also, of course, possible that even with different world states, all the disjuncts could not be represented because of contradictions encountered in the world state. In this case, depending upon the nature of the contradictions (CK-LOG will present to the user the reasons for such contradictions), one may have to abandon the ASSERT attempt (i.e., the ASSERT will fail). In this case, if the ASSERT statement was on left side of the sequent, the goal and the sequent itself will have to be both abandoned, and one has to backtrack to the parent sequent of the abandoned sequent. If the ASSERT was on the right side, one may proceed with another goal on the right side, if there was one.

The next interesting case occurs for the variable, ug-x. This is a universal variable which ranges over the (bases-of White). The condition associated with ug-x should be satisfied for *every base*, ug-x, that can be found. This condition in the ASSERT statement is an implication. The application of [ASRT-IMP →] rule to this implication, and subsequent application of [ASRT-DNF] and [ASRT-OR →] rules, will result in the following four sequents:

[ASSERT (..., (NOT (ug-x in (bases-of White))), ...)] → ;
[ASSERT (..., (isin ug-x Kodiac-Island), ...)] → .. ;
[ASSERT (..., (NOT (isin ug-x Aleutian-Island)), (NOT (isin ug-x Alaska)), ...)] → .. ;
[ASSERT (..., ((between ..)(destroyed-by ug-x Green), ...)] → .. .

For bases that are not bases of White, the first sequent above will be satisfied, and for the base in Kodiac-island, the second one will be satisfied. For those not in Aleutian-Island or Alaska, the third one will be satisfied, and for those not satisfying the first three, the fourth one will be asserted, if not already satisfied.

If none of White's bases exist in the world state, then the above sequents will cause CK-LOG to prompt the user to specify possible bindings for ug-x. To represent the bases specified by the user, new instances of BASE will be created. The creation of these new instances will automatically cause its associated *prompt* relations also to be defined. For each of the created bases, one or more of the above sequents will be made true. Thus, after this ASSERT process, for each one of the four sequents above, there will be bases in the world state for which they are satisfied. Also, for every one of White's bases in the world state at least one of the above four sequents will be satisfied.

Note that assertion of '(destroyed-by ..)' relation will in all cases also set the status of the base to destroyed. This will happen because of the restriction,

[(destroyed-by base force) IMPLIES (status-of base destroyed)],

that is associated with the structure, (destroyed-by BASE FORCE).

The remaining conjuncts in the ASSERT of seq-8 will get similarly analyzed and made true in the world state. Once this is done, the ASSERT statements in each sequent will get replaced by the conjunction that appears as its argument and the bindings given to the variables in the statements will be recorded. Also, the variables ei-t and ei-t1 in Table 1 will be set to the time at which the assertion was successfully completed. Thus, ei-t will be a time in (1984 May 25), assuming that the user started this process soon after the commander received the operational order. This will ultimately result in seq-9 shown in Table 2, where *conjn-1* is the conjunction of the arguments of the ASSERT statement in seq-8. At this point, CK-LOG will choose to consider the next subgoal, ((after ei-t) (CREATE --)) in seq-9. The analysis of this subgoal is presented in the next subsection.

5.1.2. Generation of the Planning Alternatives

The initial sequence of deductions made on the ((after ei-t)(CREATE --)) statement is shown in Table 3. In our case here, the assimilation of the ASSERT statement in the world state did not cause multiple world states with different contexts. The currently existing world state is the only state in which the creation of RadmWB-Objective needs to be analyzed. Thus, seq-9 is chosen as the starting point for the analysis. If multiple world states had occurred, then this creation would have been done in each world state, resulting possibly in reaching different conclusions in the different worlds.* For convenience, the RadmWB-Objective is reproduced below together with the abbreviations used in Table 3:

RadmWB-Objective = (*dur-term* (PREVENT *exp-1*))
dur-term = (during [(1984 June) (1984 Sept)])
exp-1 = ((EXISTS base (bases-at Attu-Island)) *conjn*)
conjn = ([belongs-to base Green] AND
[type-of base advanced] AND
[status-of base operational])

The rule [CRR-TM] in Table 3 moves the '(after ei-t)' time expression inside the CREATE expression, and [TMR-XN] combines it with the *dur-term* using **tmxn** resulting in seq-10. The application of the rules in ([CRR-OP2], [PRR-INT], [→ AND]) in the left to right order to seq-10 results in seq-11a and seq-11b. Suppose that the user chose to focus on the DESTROY expression in seq-11a (this is marked with [#] in Table 3). The rules ([DES-TM], [TMR-E]) move the time condition into the scope of the existential quantifier in *exp-1* resulting in seq-12. Application of ([→ DES-E], [DES-AND], [→ OR]) to seq-12 will result in seq-13. This sequent says that the goal may be achieved in one of two ways: either by destroying the existence of eg-x1 in Attu-Island before [1984 June], or by destroying the properties of eg-x1 in the *conjn* before the same time. Let us focus on the [#]ed expression in seq-13.

*This will happen, for example, in cases where planning is done with *assumptions* postulated by the user. In this case, the user is required to plan both for the case where the assumptions are true and for the case where it is false (see NWP-11 for details).

Table 3—Analysis of CREATE Subgoal

Name	Sequents and Bindings	Rules Used
seq-9	*conjn-1* → ((after ei-t)(CREATE RadmWB-Objective)) ;	[CRR-TM](1) [TMR-XN](1)
seq-10	.. → (CREATE (*dur-term* (PREVENT *exp-1*))) ; (**tmxn** ((after ei-t) (*dur-term* ..))) = (*dur-term* ..).	[CRR-OP2](1) [PRR-INT](1) [→ AND](1)
seq-11a	.. → ((before [1984 June]) (DESTROY *exp-1*))[#] ;	[DES-TM] [TMR-E]
seq-11b	.. → ((before [1984 Sept]) (PREVENT (CREATE *exp-1*))) ;	-
seq-12	.. → (DESTROY ((EXISTS base ((before [1984 June]) (bases-at Attu-Island))) ((before [1984 June]) *conjn*)))[#] ;	[→ DES-E] [DES-AND] [→ OR]
seq-13	.., → (DESTROY ((before [1984 June])(eg-x1 in (bases-at Attu-Island))))[#], (DESTROY ((before [1984 June]) (**subst** eg-x1 base *conjn*))) ;	- [DES-EL1] -

CK-LOG will first try [DES-EL1] and find that (bases-at Attu-Island) has no value in the world state, because it does not know of any bases at this island. Since it knows that (bases-at REGION) = BASE, it will now attempt to find all the bases in the world state that satisfy *exp-1*. In this process, it will identify all the advanced operational bases of Green in the Northern-Pacific region, and keep these as the candidates bases which may be assigned as values for the (bases-at Attu-Island)* term, since at this point CK-LOG does not know that bases at a place cannot be simply assigned to another place. CK-LOG will present these candidates to the user, with the query:

(eg-x1 in (bases-at Attu-Island),
Range for eg-x1: {'list of advanced operational bases of Green'},
eg-x1 = ?

When the user reponds with NIL, indicating that the value for (bases-at Attu-Island) is NIL, it will simply replace the DESTROY expression with '(NOT (eg-x1 in (bases-in Attu-Island)))': there are no bases in Attu-Island to be destroyed. Since eg-x1 has no possible bindings, the second DESTROY expression in seq-13 will also be false in the world state. Thus, this will also get replaced by '(NOT ...).' This indicates that the DESTROY goal is already satisfied.

Note that answers to queries like the one raised above may not be always readily available. It may be necessary for the planner to seek intelligence information to answer the questions. We will later encounter a situation where this happens.

*As it turns out, this is quite a useful thing to do in any case. As a result of this, the user will get a listing of all the advanced operational bases of Green in the Northern-Pacific region.

At this point, the focus of attention will transfer to the (PREVENT (CREATE --)) expression in seq-11b. The sequence of deduction on this expression is shown in Table 4.

Table 4—Analysis seq-11b of Table 3

Name	Sequents and Bindings	Rules Used
seq-11b	.., → ((before [1984 Sept])(PREVENT (CREATE *exp-1*)))[#] ;	[PRR-AB]
seq-13	.., → ([(DESTROY ((before [1984 Sept])(CREATE *exp-1*)) AND (PREVENT (CREATE (CREATE ((before [1984 Sept]) *exp-1*)))]	[→ AND] [CRR-OP]
seq-14	.., → ([(DESTROY ((before [1984 Sept])(CREATE *exp-1*))[#] ;	-
seq-15	.., → [PREVENT (CREATE ((before [1984 Sept]) *exp-1*)))])[#] ;	-

Applying the [PRR-AB] to seq-11b results in seq-13. Now applying [→ AND] will split seq-13 into two sequents, seq-14 and seq-15, where seq-15 is obtained after the application of [CRR-OP], which eliminates the nested CREATE expression in seq-13. These two sequents represent the two tasks, both of which require planning. Let us choose to analyze the DESTROY expression in seq-14.* The sequence of inferences on this sequent is shown in Table 5. Applying [CRR-TM] and [TMR-E] to the [#]ed expression in seq-14 will move the time condition inside the scope of the existential quantifier in *exp-1*, modifying it to,

((EXISTS base ((before [1984 Sept])(bases-at Attu-Island)))
((before [1984 Sept]) *conjn*)).

At this point [CRR-E] is applied, resulting in seq-16, where

((before [1984 Sept])(**subst** eg-x2 base *conjn*)) =
((before [1984 Sept])
([belongs-to eg-x2 Green] AND
[type-of eg-x2 advanced] AND
[status-of eg-x2 operational])

Applying [TMR-AND] to this conjunction will transform it to

([(before [1984 Sept])(belongs-to eg-x2 Green)] AND
[(before [1984 Sept])(type-of eg-x2 advanced)] AND
[(before [1984 Sept])(status-of eg-x2 operational)]),

*It may be noted that seq-15 will recursively generate again the combination represented by the sequents 14 and 15. Thus, seq-14 is the only choice that now remains to be explored. At this point, I am not claiming that the system would notice this automatically and propose to the user the choice seq-14 as the only next goal to be analyzed. But it should not be difficult to introduce this capability into CK-LOG.

Table 5—Analysis of seq-14 of Table 4

Name	Sequents and Bindings	Rules Used
seq-14	.., → (DESTROY ((before [1984 Sept]) (CREATE *exp-1*)))[#] ;	[CRR-TM] [TMR-E] [→ CRR-E]
seq-16	.., → (DESTROY (CREATE ([(before [1984 Sept])(eg-x2 in (bases-at Attu-Island)] AND [(before [1984 Sept]) (**subst** eg-x2 base *conjn*)]))))[#]	[CRR-AND](1,1) [TMR-AND] [ASRT-ASRT] [DES-ASRT] [DES-AND] [→ OR]
seq-17 (1) (2) (3) (4)	.., → (DESTROY (CREATE ((before ..])(eg-x2 in (bases-at ..)))))), (DESTROY (CREATE ((before ..)(belongs-to eg-x2 Green))))[#], (DESTROY (CREATE ((before ..)(type-of eg-x2 advanced)))), (DESTROY (CREATE ((before ..)(status-of eg-x2 operational))))]) ;	-

moving the time condition to the respective conjuncts. Now applying [CRR-AND] to the CREATE expression in seq-16 will replace the CREATE expression by:

(ASSERT
((CREATE ([before [1984 Sept])(eg-x2 in (bases-at Attu-Island)))) AND
(CREATE
([(before [1984 Sept])(belongs-to eg-x2 Green)] AND
[(before [1984 Sept])(type-of eg-x2 advanced)] AND
[(before [1984 Sept])(status-of eg-x2 operational)])))).

[CRR-AND] is now applied to the second CREATE expression above, resulting in

(ASSERT
(CREATE ([before [1984 Sept])(eg-x2 in (bases-at Attu-Island)))),
(ASSERT
([CREATE ((before [1984 Sept])(belongs-to eg-x2 Green))],
[CREATE ((before [1984 Sept])(type-of eg-x2 advanced))],
[CREATE ((before [1984 Sept])(status-of eg-x2 operational))])))

The application of [ASRT-ASRT] will then remove the nested ASSERT in the above expression, and then [DES-ASRT] will eliminate the ASSERT altogether from the DESTROY expression in seq-16, resulting in:

(DESTROY
([CREATE ([before [1984 Sept])(eg-x2 in (bases-at Attu-Island)))] AND
[CREATE ((before [1984 Sept])(belongs-to eg-x2 Green)] AND
[CREATE ((before [1984 Sept])(type-of eg-x2 advanced)] AND
[CREATE ((before [1984 Sept])(status-of eg-x2 operational)])

Finally, [DES-AND] and then [→ OR] will transform the above to the one shown in seq-17 of Table 5. This sequent may be interpreted as follows:

There are four possible ways of achieving the goal of seq-14; namely, the goal of destroying the creation of an advanced operational base at Attu-Island before [1984 Sept]:

1. Destroy the creation of the base eg-x2 in Attu-Island, or
2. Destroy that it belongs to Green, after creation, or
3. Destroy that it is an advanced base, after creation, or
4. Destroy its operational status, after creation.

Any one of these will satisfy the goal. These are the available own courses of action for the commander. The user has to explore all the four possibilities in order to choose the one that is appropriate. Let me choose to explore first the second goal above, represented by the [#]ed expression in seq-17, and take this opportunity to illustrate how the RadmWB-Purpose (discussed in Section 4.3.1) may used to reduce the number of choices.

5.1.3. Limiting the Scope of One's Objectives

The RadmWB-Purpose is the same as AdmWA-Objective; namely, to take control of Attu-Island away from Green. The formal statement of this purpose is reproduced below:

RadmWB-Purpose: [CREATE ((after [1984 Sept])(controls Attu-Island White))]

Note that this objective and the amphi-assault that has been planned to achieve this objective, have both been already introduced into the current world state. To consider this objective in the planning process, the user will now introduce the above statement as a new expression (this is a hypothesis) on the left side of seq-17.* The modified seq-17 is shown in Table 6 as seq-18. The assumption here is that a base, eg-x2, has already been created on Attu-Island, but as of [1984 June], this base did not exist.

To tell the system that the base did exist (before [1984 Sept]), the user will create a new world state context, one that is associated with seq-18, and perform in this world state the following action:

((before [1984 Sept])
(CRI eg-x2 BASE
(belongs-to eg-x2 Green)
(bases-at Attu-Island eg-x2))).

The deductions in Table 6 occur in this new world state.

Since the base eg-x2 belongs to Green before [1984 Sept], the rule [CREL-1] will simply replace the CREATE expression on the right side of seq-18 by its argument. Now, application of [DES-EL2] will cause CK-LOG to fetch the actions needed to destroy this property. This is done by the evaluation of the **destroy-action-of** function. The rule for destroying

'((before [1984 Sept]) (belongs-to eg-x2 Green)),'

that is now invoked by the **destroy-action-of** function in seq-19, is shown in Table 7 below. The rule is again stated as an inference rule. It is explicitly associated with the pattern (belongs-to BASE FORCE). This rule may be paraphrased as follows: there should exist a force, x, which does not currently control the region of the base, and which takes control of the region of the base before the specified time. In

*In fact, it is good practice to start the planning process with this hypothesis on the left side of the very first goal sequent.

Table 6—Deductions from Modified Seq-17 of Table 5

Name	Sequents and Bindings	Rules Used
seq-18	.., (CREATE ((after [1984 Sept])(controls Attu-Island White))) → (DESTROY (CREATE ((before [1984 Sept]) (belongs-to eg-x2 Green))))[#], --- ;	[CREL-1] [DES-EL2]
seq-19	.., (CREATE ((after [1984 Sept])(controls Attu-Island White))) → **(destroy-action-of** ((before ..)(belongs-to ...))), --- ;	-
seq-20	.., (CREATE ((after [1984 Sept])(controls Attu-Island White))) → ((EXISTS x FORCE) ([NOT (controls Attu-Island x)] AND [CREATE ((before [1984 Sept])(controls Attu-Island x))])), --- ;	[→ E] [→ AND] [→ N]
seq-21a	.., (CREATE ((after [1984 Sept])(controls Attu-Island White))) → (eg-x3 in FORCE) ;	-
seq-21b	.., (CREATE ((after [1984 Sept])(controls Attu-Island White))) (controls Attu-Island eg-x3) → ;	
seq-21c	.., (CREATE ((after [1984 Sept])(controls Attu-Island White))) → [CREATE ((before [1984 Sept])(controls Attu-Island eg-x2))] ;	

Table 7—*Destroy-action* for (belongs-to BASE FORCE)

Condition	Inference
T	((EXISTS x FORCE) ([NOT (controls (base-isin base) x) AND [CREATE (tm-term (controls (base-isin base) x)]))
	(tm-term (belongs-to base force))

seq-19, this rule is invoked for force = Green, base = eg-x2, and tm-term = (before [1984 Sept]). The result of this invocation is shown in seq-20 of Table 6. Notice that during the invocation process, the term '(base-isin base)' that appears in Table 7 gets replaced by Attu-Island, since this is the place of the base eg-x2.

Now applying [→ E] to seq-20 will eliminate the existential quantifier in the sequent generating the new variable eg-x3 which is substituted for x in the conjunction of seq-20. Subsequently, [→ AND] eliminates this conjunction causing seq-20 to split into three sequents, seq-21a through seq-21c. Seq-21b in Table 6 is obtained by applying [→ N] to the (NOT ...) expression after the split.

At this point, the user may notice the redundant goals in seq-21c: the plan to CREATE the take-over of control of Attu-Island after [1984 Sept] has been already hypothesized. Thus, the goal to take control of it before [1984 Sept] would in effect cause RadmWB to try to achieve his superior's goal ahead of time. This will cause the user to abandon this line of inquiry. Notice that the system may at best only point out the similarity between the two create goals in seq-21c, but could not itself suggest that this goal should be abandoned because of the similarity. This is a decision for the user to make.

In this case, the redundant subgoal turned out to be similar to a superior commander's goal. In a large planning activity, it is quite possible that there are commanders A and B who in their planning activities choose similar subgoals. By making the planning subgoals of one commander available to another, the system may be used to point out similar goals and thus help the commanders to suitably limit their objectives. As the above discussion illustrates, the user actively participates in the analysis process by choosing subgoals, making assumptions, and interpreting the results of the analysis.

The user will now revert back to the previous world state, the one associated with seq-17. The system will save the world state associated with seq-18 for possible future use, unless the user advises the system to forget it. The user may now modify seq-17 by deleting from this sequent the [#]ed expression, since this is not a line of inquiry that he wishes to undertake. The modified sequent, seq-22, is shown below and in Table 9.

seq-22 .., →
(1) (DESTROY (CREATE ((before ..])(eg-x2 in (bases-at ..)))))[#],
(3) (DESTROY (CREATE ((before ..)(type-of eg-x2 advanced)))),
(4) (DESTROY (CREATE ((before ..)(status-of eg-x2 operational)))) ;

This sequent presents to the user only three alternatives; namely, the goals (1), (3), and (4). These are the current candidate courses of actions that the planner has to analyze. Let us now take a look at what CK-LOG has achieved so far.

5.1.4. Comments on Mission Understanding

So far, CK-LOG has not significantly helped the user in his planning tasks. The kinds of conclusions that the system has arrived at are rather obvious ones to any human planner. With the help of the user, CK-LOG has reduced the mission to its three essential components. It presented to the user the opportunity to notice that the mission should not be accomplished by taking control of Attu-Island, because this is the superior commander's objective.

The purpose of presenting this analysis here is to show how CK-LOG unravels this understanding starting from the very general goal, (ACHIEVE (*design* opplan)). CK-LOG's reasoning processes do indeed exhibit the quality of understanding for the mission that a human would expect of a reasonable system. We see here that within the general reasoning paradigm of CK-LOG this kind of mission understanding occurs naturally. The initial progression of analysis presented in this section is a necessary part for what is yet to follow. The important point to note is that the methods presented above were in no way specific to the mission in our example. The same method of analysis would apply to any mission statement in the language DL. The ability of CK-LOG to analyze a given mission will be limited only by the knowledge it has of the actions that are involved in the realization of the objective. This is a part of the general knowledge of the Operational Planning domain.

During this reasoning process, CK-LOG has also identified some of the possible physical objectives relevant to this planning: the bindings that were so far used for the various variables during the reasoning process provide the candidate physical objectives. As of now, these include all the bases of Green in the North-Pacific region, and also, of course, Attu-Island itself. Of course, this is too broad a category of items. But at this point of our reasoning, we could not have done any better. We do not yet have all the relevant information, and we have not yet analyzed any of the possible military actions.

Again, it is important to note that, choosing the bindings as candidate physical objectives will guarantee *that none of the objects that are relevant to the accomplishment of a given mission will be missed.* Once the commander's decision is made, this list of candidate physical objectives can be drastically reduced. But until this is done, having a broad category of possible objectives will help assure that none of the significant alternatives are missed.

The performance of CK-LOG discussed above shows clearly that it is capable of correctly understanding the commander's mission. As we will see below, further analysis of the [#]ed goal in seq-22 will cause the system to seek from the user additional information concerning a possible enemy plan to build a base at Attu-Island, and raise the relevant additional questions concerning the base building operation. Ultimately, this leads the user to arrive at the *commander's decision*, which in this case turns out to be a decision to destroy the convoy that the enemy is planning to use to transport the base building materials to the island.

5.2. The Making of the Commander's Decision

The commander's decision is arrived at through the analysis of goal (1) in seq-22. Let me first outline the reasons why CK-LOG would lead the user to reject the goals (3) and (4) in preference to goal (1). The analysis of (3) and (4) would present to the user courses of actions that he might follow if his principal objective, namely, the one that goal (1) in seq-22 leads to, does not succeed. Without presenting the details, let me first summarize this analysis before presenting the deductions from goal (1).

Alternatives (3) and (4) assume that the base has been already built at Attu-Island.* In order to destroy the operational status of this base, or its advanced status, White would be called upon to mount an air and sea attack on this base with appropriate choices of physical objectives. These are the actions associated with the destruction of the relations, '(type-of base advanced),' and '(status-of base operational).' The system's knowledge base of an advanced base would indicate that it would be well-defended (i.e., the base will be within the well-defended region of the enemy). After the base is built at Attu-Island, the entire island will move into the well-defended region of Green. Before this, the island was outside the region defended by Green.

A general property of any attack on a well-defended region is that the risk factor for the attacking force loosing its own forces is high. As mentioned in Section 2.1.1, the risk factor is one of the properties of MIL-ACTION, described by the 'risk-factor-of' relation. The restriction associated with the risk-factor-of relation will specify that if the region of a military action was well-defended by the enemy, then the risk factor would be high, and if the region was not defended at all, then the risk factor would be low.

Thus, the required air and sea attack actions would acquire a high risk factor. But, if the action was the prevention of the construction of the base, since before the construction of the base Attu-Island was outside the region defended by Green, the system would identify military actions with low risk factors which could accomplish the prevention goal. Thus, a comparison of the military actions resulting from the analysis of (1), (3), and (4) would lead to the selection of (1) as the preferred alternative.

The analysis of goal (1) is presented in Table 9. When [CREL-1] is applied to the [#]ed expression in seq-22 of Table 9, CK-LOG will find out that the argument of the CREATE expression is not true in the current world state (since it is known that (bases-at Attu-Island) is NIL). Thus, [CREL-2] will be applied to this CREATE expression. This rule will fetch the *create-action* for '(eg-x2 in (bases-at Attu-Island)).' The *create-action* for '(eg-x2 in (bases-at Attu-Island)),' is the same as the one for '(bases-at Attu-Island eg-x2).'† This definition is associated with the structure, (bases-at REGION BASE), and is shown in Table 8.

*Without the base being there, it could not have the properties indicated in goals (3) and (4).

†The range definition, expressed by the 'in' relation, is here converted to a form that directly refers to the properties of objects involved in the relation. The range definitions are useful for evaluating the truth values of logical expressions in a world state. Their conversion to equivalent relational forms may be controlled by inference rules defined for them. To simplify discussion I have here assumed that the conversion is done by the access functions themselves.

Table 8—*Create-action* for (bases-at REGION BASE)

Condition	Inference
[NOT (OCCURS base))]	[(EXISTS x BUILD-BASE) ((agent-of x (controls region)) AND (base-of x base) AND (region-of x region) AND (starting-time-of x (before (MINUS-TIME tm-term (needed-time-for x)))) AND (ending-time-of x tm-term))]
	(tm-term (bases-at region base))

The definition says in effect that to put a base at a place, it has to be built there, and the base building process can be invoked only for bases that do not already exist in the world state. Also that the base building action, x, should be started at a time that is at least (needed-time-for x) before the time specified by the tm-term.* This needed time may, of course, itself depend on the base that is to be constructed, and the region where it is to be built. This rule is now invoked for region = Attu-Island, base = eg-x2, and tm-term = (before [1984 Sept]). The result of this invocation is shown in seq-23 of Table 9.

During the invocation of this rule, the access function evaluates '(controls Attu-Island)' to Green, but keeps '(before (MINUS-TIME (before [1984 Sept]) (needed-time-for x)))' as it is because x does not as yet exist in the world state and thus the (needed-time-for x) is as yet unknown. The important point to note here is that while invoking this create definition, the terms that appear inside the expressions of the definition are evaluated with respect to the current world state, and the values of these terms in the world state are substituted for the terms themselves.

The first rule that CK-LOG will attempt on seq-23 is the [DES-EL1] rule which checks whether the argument of the DESTROY expression is false in the world state. Our world state does not have in it any base building actions. Thus, the possible bindings for bb1 are currently unknown. The other elimination rule, [DES-EL2], requires that the argument should be a true positive atomic expression. Thus, it is not applicable. This will cause CK-LOG to prompt the user about the base building operations that satisfy the given expression, before it proceeds to apply the other applicable rules for the DESTROY expression. The following question is thus presented to the user:

[(EXISTS x BUILD-BASE)
((agent-of x Green) AND (base-of x eg-x2) AND
(region-of x Attu-Island) AND
(starting-time-of x (before (MINUS-TIME (before [1984 Sept]) (needed-time-for x)))) AND
(ending-time-of x (before [1984 Sept 1])))])] ?

In effect, CK-LOG is here asking the user whether he is aware of any base building operations at Attu-Island by Green with the indicated properties. It so happens, in this example in NWP-11, RadmWB does

*The function, MINUS-TIME, in Table 8 is one of the system functions of CK-LOG. It can take as arguments either time instants or tm-terms. If its arguments are unbound, then it will return (a possibly reduced form of) the MINUS-TIME expression itself.

Table 9—Analysis of Modified Seq-17 of Table 6

Name	Sequents and Bindings	Rules Used
seq-22 (1) (3) (4)	.., → (DESTROY (CREATE ((before ..])(eg-x2 in (bases-at ..)))))[#], (DESTROY (CREATE ((before ..)(type-of eg-x2 advanced)))), (DESTROY (CREATE ((before ..)(status-of eg-x2 operational)))) ;	[CREL-2]
seq-23	--, → (DESTROY [(EXISTS x BUILD-BASE) ((agent-of x Green) AND (base-of x eg-x2) AND (region-of x Attu-Island) AND (starting-time-of x (before (MINUS-TIME (before [1984 Sept]) (needed-time-for x)))) (ending-time-of x (before [1984 Sept 1]))])], -- ;	[DES-EL1] [DES-RED]
seq-24	.., → (DESTROY [(bb1 in BUILD-BASE) AND (agent-of bb1 Green) AND (base-of bb1 Attu-base) AND (region-of bb1 Attu-Island) AND (ending-time-of bb1 (before [1984 Sept])) AND (starting-time-of bb1 (before [1984 Aug 17]))], -- ;	[DES-AND] [→ OR]
seq-25	.., → [DES-EL2](6) (DESTROY (bb1 in BUILD-BASE), (DESTROY (agent-of bb1 Green)), (DESTROY (base-of bb1 eg-x2)), (DESTROY (region-of bb1 Attu-Island)) (DESTROY (starting-time-of bb1 (before [1984 Aug 17])) (DESTROY (ending-time-of bb1 (before [1984 Sept 1]))), -- ;	

receive intelligence information from AdmWA telling him that Green does have plans to build a base at Attu before [1984 Sept]. Indeed, it was the knowledge of this intelligence that caused AdmWA to issue the said oporder to RadmWB.

At this point, if the user did not have the information to answer the above question, then he might do one of two things: he might hypothesize that such a base building operation will occur, introduce this hypothesis into the world state with an ASSERT statement, and introduce the results of this assertion on the left side of the current sequent as hypotheses, and continue with his planning. Or, he might decide to actively look for the needed intelligence information. Here is a simple case, where CK-LOG prompts the user to confirm possible enemy operations or objectives that are germane to the current planning task. This is typical of the kinds of prompting that occur in CK-LOG during a planning activity, because of incomplete knowledge in the current world state.

The user may now respond to the above question with the following assertion*

ASRT-1: (ASSERT
((EXISTS x BUILD-BASE)
(agent-of x Green) AND (base-of x Attu-base)
(region-of x Attu-Island) AND
(ending-time-of x (before [1984 Sept]))) → ;

Since this assertion appears to the left of →, it will now be assimilated into the current world state. This assimilation process will cause CK-LOG to create a BUILD-BASE operation with the properties specified in the above assertion. Let bb1 be the name of this new operation. The above assertion will also cause CK-LOG to create Attu-base as a new instance of a BASE, noting that this is under construction (it is the base of bb1). The restriction on (base-isat BASE) will specify that its value should be the same as the region-of the BUILD-BASE action, bb1; namely, Attu-Island. Its precise location, which is the (location-of Attu-base), in the island will remain unknown, along with all of its other properties. The above assertion leaves out the starting time of the base building operation, bb1.

After assimilating the assertion, CK-LOG will proceed to reevaluate the argument of the DESTROY expression in the new world state. This will cause CK-LOG to assign bb1 as a possible value for the existentially quantified variable, x, in the DESTROY expression, and return the user the following **true-part-of** the expression:

[(bb1 in BUILD-BASE) AND (agent-of bb1 Green) AND
(base-of bb1 Attu-base) AND (region-of bb1 Attu-Island) AND
(ending-time-of bb1 (before [1984 Sept])) AND (starting-time-of bb1 (before [1984 Aug 17]))].

The above expression will now be substituted into seq-22, by using the DESTROY reduction rule [DES-RED], as the new argument for the DESTROY expression, resulting in seq-23. Notice that CK-LOG had assigned a starting time for bb1. This happens because of the following:

In the BUILD-BASE action that was defined in Section 2.5, the needed-time-for building a base was set to 2 weeks (this time was chosen quite arbitrarily). Whatever this time is, it is a quantity that should be determinable from the nature of the base and where it is being built. If this time was unknown and was not automatically determinable, then the user will be at this time prompted for it. For our purposes here, let us assume that 2 weeks is a reasonable amount of time for building a base under war conditions. Under these circumstances, the MINUS-TIME expression

(MINUS-TIME (before [1984 Sept]) [WEEKS 2])

will evaluate to '(before [1984 Aug 17]))' since by default [1984 Sept] is interpreted as [1984 Sept 1]. Thus, the expression, '(before (MINUS-TIME ...))' in the existentially quantified expression in ASRT-1 above, will evaluate to '(before (before [1984 Aug 17]))', which is the same as (before [1984 Aug 17])). The precise starting time will still remain unknown. Finally, the application of ([DES-AND], [→ OR]) rules to seq-24 will result in seq-25 of Table 9.

The arguments of all the DESTROY expressions in seq-25 are now true in the world state. The objective of goal (1) of seq-22 may be realized by successfully accomplishing any one of the DESTROY operations in seq-25: either the base building operation, bb1, may be destroyed, or the Attu-base that is

*Since the user knows this to be a fact, this is not at this point considered to be a hypothesis, and thus the current sequent itself is not modified.

being built may be destroyed, the region-of bb1 may be destroyed, its agent may be destroyed, its starting time may be destroyed, or its ending time may be destroyed. CK-LOG will now retrieve from its knowledge base the destroy-actions for these, as a result of applying the rule [DES-EL2] the necessary number of times. This in effect will cause CK-LOG to analyze the base building operation and identify the means for destroying this operation. This is discussed in the next subsection.

5.2.1. Analysis of the Base Building Operation

The actions to destroy a base building operation and the various properties of this operation mentioned in seq-25 are shown in Table 10. The actions to destroy a build-base operation is the same as those needed to destroy (region-of build-base region) or (agent-of build-base force). I have combined the rules for these into one statement in X. The rule says that to destroy any of these, one has to destroy the *behavior* of the build-base action. To destroy (starting-time-of build-base **t-exp**), one has to destroy all the resources that are needed for the base building operation. To destroy (ending-time-of build-base **t-exp**) or (base-of build-base base), one has to destroy the resources or the installations-in the base building operation. To destroy the build-base operation from being successful, one has to destroy its installations before its ending time.

Table 10—*Destroy-actions* for BUILD-BASE

Condition	Inference
T	(DESTROY (*behavior* build-base))
	build-base (region-of build-base region) (agent-of build-base force)
T	(DESTROY ((before (starting-time-of build-base)) ((EXISTS x (needed-resources-for build-base))))
	(starting-time-of build-base **t-exp**)
T	[(DESTROY ((before (ending-time-of build-base)) ((EXISTS x (installations-in build-base)))) OR (DESTROY ((before (ending-time-of build-base)) ((EXISTS y (needed-resources-for build-base))))]
	(ending-time-of build-base **t-exp**) (base-of build-base base)
T	((EXISTS x (installations-in build-base)) ((before (ending-time-of build-base)) (DESTROY x)))
	(status-of build-base successful)

The result of invocation of the destroy actions for the atomic expressions appearing as arguments of the DESTROY expressions in seq-25 is shown in seq-26 of Table 11. The action for destroying (bb1 in BUILD-BASE) is the same as that needed to destroy bb1 itself. Note that *behavior* of a BUILD-BASE operation was defined in Section 2.5. The application of [DES-TRM] and [→ OR] rules to seq-26 results in seq-27. The rule [DES-TRM] retrieves from the knowledge base the behavior of bb1 and the application of [→ OR] replaces the OR's by commas.

Table 11—Destruction of the Base Building Operation

Name	Sequents and Bindings	Rules Used
seq-26	.., → (DESTROY (*behavior* bb1)), (DESTROY ((before (starting-time-of bb1))((EVERY x (needed-resources-for bb1)))), [(DESTROY ((before (ending-time-of bb1))((EXISTS x (installations-in bb1)))) OR (DESTROY ((before (ending-time-of bb1))((EXISTS y (needed-resources-for bb1))))] ;	[DES-TRM] [→ OR]
seq-27	.., → (DESTROY ([(EXISTS x (needed-resources-for bb1)) ((before (starting-time-of bb1)) (CREATE (location-of x (location-of bb1))))] AND [(EVERY x (installations-in bb1)) ((after (starting-time-of bb1))(OCCUR (CREATE x)))] AND [(after (*time-of* ((EVERY x (installations-in bb1))(OCCUR (CREATE x))))) (ASSERT (status-of build-base successful))])) (DESTROY ((before (starting-time-of bb1))((EVERY x (needed-resources-for bb1)))), (DESTROY ((before (ending-time-of bb1))((EXISTS x (installations-in bb1)))), (DESTROY ((before (ending-time-of bb1))((EXISTS y (needed-resources-for bb1)))) ;	[DES-AND] [→ OR] [→ DES-E] [→ DES-U] [→ DES-U] [→ DES-E] [→ DES-E] [→ AND]

Appropriate applications of [→ DES-E], [→ DES-U], [DES-AND], [→ AND], and [→ OR] rules to seq-27 now results in sequents 28a through 28e of Table 12. Each of these present four alternatives, of which the last three are the same in all of the sequents. The sequents together specify four alternatives for destroying the needed-resources-of bb1; for example, there are four possible objectives one could choose for destroying the (needed-resources-of bb1): the objective [1a] calls for the destruction of (ui-c8 in (needed-resources-for bb1)). Ui-c8 here is a universal instantiation variable. Thus, this calls for the destruction of every possible binding one could get for ui-c8. The objective [2a] calls for the destruction of the same before the ending-time-of bb1 and the objective [3a] calls for the destruction of *some* of the needed resources of bb1 (this is indicated by the use of the existential variable eg-x9) before its starting time. Objective [1b] calls for the destruction of the processes for putting the needed resources at the location of bb1. The choice of any one of these will cause CK-LOG to raise questions about what the needed resources are and where they are (because to destroy any object one has to know where it is). Similarly, there are three possible objectives, [4a], [1c], and [1d], for the destruction of the installations in bb1: destroying all of them before the ending time of bb1, destroying some of them, or destroying their creation after the starting time of bb1. Finally, the objectives [1e] calls for the destruction of the success of bb1.

In any case, bb1, its installations and the needed resources (as represented by the possible bindings for ui-c8, ui-c9, ui-c10, eg-x8, and eg-x9) will all now appear in the list of physical objectives that are relevant to the current planning task. Note that CK-LOG has generated all the possible cases for

Table 12—Destruction of Base Building (Continued)

Name	Sequents and Bindings
seq-28a	.. → [1a](DESTROY (ui-c8 in (needed-resources-for bb1))), [2a](DESTROY ((before (ending-time-of bb1)) (ui-c10 in (needed-resources-for bb1)))), [3a](DESTROY ((before (starting-time-of bb1)) (eg-x9 in (needed-resources-for bb1)))), [4a](DESTROY ((before (ending-time-of bb1)) (ui-c9 in (installations-in bb1)))) ;
Seq-28b	.. → [1b](DESTROY ((before (starting-time-of bb1)) (CREATE (location-of ui-c8 (location-of bb1))))), [2a],[3a],[4a] ;
Seq-28c	.. → [1c](DESTROY (eg-x8 in (installations-in bb1))), [2a],[3a],[4a] ;
Seq-28d	.. → [1d](DESTROY ((after (starting-time-of bb1)) (OCCUR(CREATE eg-x8)))), [2a],[3a],[4a] ;
Seq-28e	.. → [1e](DESTROY ((after (*time-of* ((EVERY x (installations-in bb1))(CREATE x)))) (ASSERT (status-of build-base successful))], [2a],[3a],[4a] ;

the destruction of bb1: to prevent bb1 from starting, to prevent it from ending successfully, to destroy the installations after they are built, or during their building process, to destroy the resources for building, or to destroy the resources from being brought to the place where building process is occurring. The user may now explore each one of these possibilities.

The following information is currently unknown in the world state: (starting-time-of bb1), (ending-time-of bb1), (installations-in bb1), (location-of bb1), and (location-of (needed-resources-of bb1)). It is known, however, that the starting time is (before [1984 Aug 17]) and the ending time is (before [1984 Sept]). CK-LOG will prompt the user for the above information. This prompting will occur when the various DESTROY expressions are chosen by the user for analysis, and the [DES-EL1] rule is invoked to check the truth of the relations to be destroyed. I will not present here the full analysis of all of these possibilities. It is worth noting that during this analysis process, CK-LOG does find opportunities to prompt the user for information that is unknown in its world state and is needed for continuing the planning process. Briefly, here is the main sequence of analysis that leads to the selection of the physical objectives for the military action and to the commander's decision.

In the NWP-11 example, the commander, RadmWB, receives intelligence information that the (needed-resources-for bb1) are not in Attu-island, but are in the port of Petropovlsk, and Green is planning to transport them to Attu-Island some time in [1984 July]. Thus, at this point the user presents to CK-LOG the following information: that all the needed resources of bb1 are at Petropovlsk, and Green has put together a convoy to transport these resources to Attu using the convoy, sometime in [1984 July]. The formal statement of this information is shown in Fig. 5.2. It makes use of the concepts CONVOY (a specialization of FORCE) and TRANSPORT (a specialization of MIL-ACTION).

If the information about Green's plans to transport the needed resources to Attu had not been known to the user, and the user knew that the resources were at Petropovlsk, then analysis of '(CREATE

```
[ASSERT
 ((EVERY x (needed-resources-for bb1))
  ((at [1984 May 25]) (location-of x Petropovlsk)) AND
  ((EXISTS convoy CONVOY)(EXISTS trans TRANSPORT)
   (belongs-to convoy Green) AND
   ((at [1984 May 25])(location-of convoy Petropovlsk)) AND
   (instruments-of trans convoy) AND (agent-of trans Green) AND
   (objects-of trans x) AND (starting-place-of trans Petropovlsk) AND
   (destination-of trans Attu-Island) AND
   (starting-time-of trans (before [1984 July 7])) AND
   (ending-time-of trans (between ([1984 July 1] [1984 July 31]))) AND
   (needed-time-for trans (WEEKS 3))] → ;
```

Fig. 5.2—Information about Green's plans

(location-of ui-c8 (location-of bb1))' in objective (4) of seq-28, would have caused CK-LOG to postulate a TRANSPORT action using a CONVOY, to transport these resources to (location-of bb1).* This is because the **create-action** for '(location-of ui-c8 (location-of bb1))' would call for such a transportation. A partial statement of this action is shown in Table 13. Essentially, it states that to put an object at a location, the TRANSPORT action should be invoked to transport the object using a CONVOY. A call to DESTROY this action would have caused CK-LOG to prompt the user to confirm the existence of such an action, which in turn would have caused him to seek the necessary information.

Table 13—*Create-action* for (location-of OBJECT LOCATION)

Condition	Inference
T	((EXISTS trans TRANSPORT)(EXISTS convoy CONVOY) (EXISTS x LOCATION) ((before tm-term)(location-of object x)) AND (CREATE ((starting-time-of trans)(location-of convoy x))) AND (belongs-to convoy (controls (region-of location))) AND (objects-of trans object) AND (destination-of trans location) AND (ending-time-of trans tm-term) AND etc.)
	(location-of object location)

The important point to note in the above analysis is that the destruction of the action is achieved by the destruction of the truth of its behavior. In the trans action discussed below, the destruction of the action is achieved by destroying the instruments-of the action. But here the behavior of the action is used to determine the time and place where the instruments should be destroyed. In most cases, the behavior of an action will be used in the analysis of its destruction and creation. In the analysis presented so far, for every DESTROY or CREATE operation, CK-LOG enumerated all the alternatives for performing the operation. It did this by analyzing the given expressions using its general reasoning abilities and reducing them to combinations of their elementary parts. For each elementary component of an expression, the appropriate actions for its creation or destruction were retrieved by CK-LOG from

*The default value for (location-of bb1) is the (region-of bb1); i.e., when the location is unknown, its region is used when necessary.

its general knowledge base of the military planning knowledge. In every case, it accounted for the relevant time constraints and prompted the user for needed information. This kind of activity forms the basis for the use of CK-LOG to reason about actions and their effects. This capability is repeatedly used in the planning process.

In the example in NWP-11, the destruction of the convoy that transports the base building resources to Attu-Island is in fact the decision made by RadmWB. This choice is preferable to the others in seq-28 because once the base building operation becomes active at Attu the defense of this operation will be in place, and this will increase the risk factor of military actions against it. In the analysis of seq-28, this will be noticed as follows: when the status of the base building operation, bb1, becomes active, the *function* of bb1 (see Fig. 2.7 in Section 2.5) will get introduced on the left side of the goal sequent, seq-28, as a hypothesis. This function specifies that Attu-Island will then be in the well-defended region of Green and actions to prevent the destruction of bb1 will be in place. This will cause the military actions to destroy bb1 to have a high risk factor.

Note that the behavior of bb1 requires that (before (starting-time-of bb1)) the creation of the needed resources of bb1 at the location of bb1 should have been successfully completed. At this point, this creation process is occurring in the current world state, but it has not been yet completed. Thus, bb1 is not yet active and the *function* of bb1 will not be on the left side of seq-28. It is now appropriate that objective (4) in seq-28 is chosen for analysis.

When the user chooses this objective for further analysis, after CK-LOG assimilates the above ASSERT statement, CK-LOG will call for the destruction of the '(CREATE (location-of ui-c8 (location-of bb1)).' This CREATE expression will cause CK-LOG to invoke the trans action mentioned above that is using the convoy to transport the base building resources to Attu-Island. Thus, the destruction of this CREATE expression will get reduced to the destruction of the trans action. This in turn will call for the destruction of the convoy of trans.

The *destroy-action* for TRANSPORT is shown in Table 14. It says that the instruments of the transportation should be destroyed between the interval, int, during which the instruments of trans are in some region, x, that is implied by the behavior of trans.

Table 14—*Destroy-action* for TRANSPORT

Condition	Inference
T	([*behavior* trans] IMPLIES [(EXISTS x REGION)(EXISTS int INTERVAL) ([(during int)(region-of (instruments-of trans) x))] AND [(between int)(DESTROY (instruments-of trans))])]
	trans

The essential behavior of trans is that (after (starting-time-of trans)) as time progresses, the convoy will approach points closer to its destination, and at its ending time, if it is successful, it will be at the destination. The formal statement of this condition and its statement in English are shown below. They are expressed in terms of expansions of the region of the destination.

> There is an expansion of the region of the destination of trans, which has the starting location of trans at its boundary. Let x be this expansion. For every expansion, y, of the destination that is within x, there will be times, t, (after (starting-time-of trans)) at which the (instruments-of trans) will be in this region y.

([(EXISTS x (expansion-of Attu-Island))
(is-at-the-boundary-of x Petropovlsk) AND
((EVERY t (after (starting-time-of trans)))
(EXISTS t1 (after t))(EXISTS t2 (after t1))
(EXISTS y (within x))(EXISTS z (within y))
([t1 (region-of convoy y)] AND [t2 (region-of convoy z)]))] AND
[(status-of trans successful) IMPLIES
((ending-time-of trans)(region-of convoy Attu-Island))])

One may think of expansions of the destination as being circular regions of progressively expanding radii centered at the destination. Note that this description does not require that the instruments of trans should forever come closer to its destination as time progresses. There may be moments during the trans, when they move away from the destination. This kind of statement makes it possible to reason about the behavior of trans without knowing precisely the route it is going to follow in its journey towards its destination. The formal description of behavior shown above has (destination-of trans) = Attu-Island, and (starting-location-of trans) = Petropovlsk.

The *function* of trans will specify that once it is active, the convoy used for the transportation itself will be well-defended: it would be defended by the forces belonging to the agent-of the trans from bases in whose defensive range the convoy is located, and also it will be defended by forces that are themselves a part of the convoy. This will, of course, increase the risk factor of attacking the convoy. There are now risks involved both in attacking bb1 after it becomes active, and in attacking the convoy after it becomes active. The commander has to choose the most favorable one. To do this, he has to proceed further with the analysis. Analysis of the destruction of trans will now reveal the options available for the times and places where the convoy might be destroyed. This analysis is quite similar to the analysis of the behavior of bb1. Without presenting the full details of this analysis, let me now summarize the conclusions.

As per details given in NWP-11, it so happens that the well-defended range of Green just falls short of Attu-Island by a few hundred miles. In this region, Green has no defenses whatsoever. The analysis of the behavior of trans will now reveal the choices to consider the times and locations where one might choose to destroy the convoy. The poorly defended region around Attu-Island, mentioned above, is an expansion of the island region. As per the behavior of trans shown above, this region is bound to be reached by the convoy some time after the starting time of trans. In the analysis, this region will appear as one of the possible bindings for the (region-of convoy). Let y be this region. The objective may now be stated as:

[(EXISTS int INTERVAL)
([(during int)(region-of convoy y))] AND
[(between int)(DESTROY convoy)])]

Thus, the user chooses to attack when the convoy comes out of the well-defended range of Green. This reduces the risk factor for his military actions, compared to his other options. The system will now prompt the user for the time when the convoy reaches this range and the location where it does this. To find this information, the commander decides to mount an intelligence gathering operation using his submarine force, and the planning proceeds further with the means to be employed to destroy the convoy and myriads of other details.

We have now reached the objective we set out for ourselves in Section 4.1.2: we have analyzed the planning processes involved in the *development of commander's estimate of the situation.* The commander has chosen the convoy as the physical objective, and the destruction of this convoy as his decision. These two choices together represent the *commander's estimate of the situation.* I have shown above how

CK-LOG would help the commander to arrive at these choices. This exercise has shown how planning knowledge is represented in CK-LOG, and how the facilities of CK-LOG are used to set up planning subgoals, identify own and enemy objectives and actions, and information concerning the times, places, and instruments of the actions, and reason about these in a planning context. It has pointed out the essential CK-LOG mechanisms that are involved in the knowledge representation and the problem solving processes of CK-LOG.

We have seen how CK-LOG identifies at each state of the planning process the relevant information that it knows, the actions and objectives that are appropriate, and how it prompts the user at the right times for information that it needs to know but does not know. We have also seen how CK-LOG has the ability to assimilate new information given by the user during the planning process, and how, at each stage of the planning process, CK-LOG presents to the user all the planning alternatives, provides to the user the opportunity to analyze all of them to the level of detail that he desires, and actively assists the planner in the analysis of the alternatives. By doing this, I have demonstrated the *descriptive adequacy* and the *problem solving adequacy* of CK-LOG. The organization of the military planning knowledge that I have presented here, and the logical reasoning abilities that CK-LOG has to use this knowledge, together provide the foundations of a mechanical system that can intelligently assist a human planner to systematically explore all planning alternatives and help organize planning activities in a logically coherent way.

This, then, is the essence of the capabilities of the OPPLAN-CONSULTANT. The organization presented here follows the guidelines established in NWP-11. The analysis shows that the system does indeed have the fundamental capabilities that are essential to function as an effective planning consultant for Naval Operational Planning.

6. CONCLUDING REMARKS

This report has introduced the principal concepts in the organization and operation of the logic-based knowledge processing system CK-LOG and has shown how it might be used to develop an expert consultation system for Naval Operational Planning. Numerous examples of descriptions of military planning knowledge in CK-LOG were presented here, and the report showed how this knowledge may be used in CK-LOG to solve military planning problems. The organization of the planning knowledge and the planning methodology are based on the guidelines given in NWP-11. The planning problem discussed here was also chosen from NWP-11. For those unfamiliar with NWP-11, this report provides a reasonably comprehensive introduction to the planning concepts discussed in that manual. The examples presented here demonstrate the *descriptive adequacy* of CK-LOG to describe, represent, and use the concepts involved in military planning. The analysis of the solutions generated by the calculus of sequents for the planning example, presented in Section 5, gives a partial demonstration of the *problem solving adequacy* of CK-LOG for solving military planning problems using the guidelines given in NWP-11. The report thus establishes the viability of using CK-LOG for the development of the OPPLAN-CONSULTANT.

The organization and operation of CK-LOG also makes several fundamental contributions to the technology of *knowledge representation systems* in AI, as discussed in Section 3.1.3. The most significant of these are the following:

- It shows a way of combining the *general theorem-proving* paradigm for stating and solving problems with the knowledge representation paradigm based on *frames*.
- It shows how models may be used in a theorem-proving process to represent and reason about *possible worlds*.

- It extends the standard notions of problem solving, using theorem-proving to a logical system that uses *modal operators* to describe *possible worlds* that ongoing actions in a world state might create.

- It extends the *frame* representation paradigm to the paradigm of CK-LOG *concepts.* It shows how the *function* and *behavior* of *concepts* can be used to reason about the behavior and effects of ongoing actions in a world state.

- It shows how special facilities for reasoning about time and actions can be incorporated into a problem solving scheme that is based on general theorem-proving.

- It shows how models in a three-valued logical system may be used to generate hypotheses during a problem solving process, based on information that is unknown in the world state, but is needed for the solution of the problem.

CK-LOG offers a powerful and compelling alternative to the *situation calculus* [17] mode of representing and reasoning about actions. CK-LOG uses the *frame*-based knowledge representation system, MDS, in two ways: one is for defining and representing the knowledge **K[U]** of a universe **U** and for using this knowledge to build models of the world states in the universe, and the other is for defining and modelling its own inference engine. The inference engine uses a theorem-proving system that is based on a variant of Gentzen's Natural Deduction System. It incorporates two principal variations: the first is the use of a new algorithm called the *mating algorithm* for testing proof terminations. The second is the use of specialized inference rules for reasoning about (*modal*) operator expressions using the *possible world* semantics.

The mating algorithm is a dual of the *unification algorithm* used in *resolution theory.* Whereas unification produces the most general substitutions, mating produces the most specific substitutions; also, where as in resolution the unified literals are removed from the *clauses*, in mating, the mated *atoms* generate *atoms* which are added to the *sequents.* The algorithm gives the theorem-proving system of CK-LOG several new capabilities: to identify information pertinent to a given problem and retrieve it from the knowledge base, to update its models of possible worlds during the problem solving process based on the findings of the theorem-proving system, to use these models to test proof terminations, and to generate hypotheses during the problem solving process. CK-LOG is not just a knowledge representation system. It is a logic-based knowledge processing system, just as PROLOG is a logic-based programming system.

CK-LOG is now being implemented in MDS.* MDS itself is implemented in Rutgers ELISP on the DEC-TOPS-20 system. As the first implementation of its kind, there is much room for improvement both in efficiency and in basic system organization. Also, there is need to gain experience on the use of CK-LOG, as a knowledge processing system, to implement expert consultation systems in complex domains. (I am now using the development of OPPLAN-CONSULTANT as an experimental vehicle.) This planning domain poses several challenging problems. It is this challenge that helped crystallize the innovations in CK-LOG. Eventually, to be effective as an expert consultant in the Naval Operational Planning domain, CK-LOG should also have some facilities for *evidential reasoning*,† much as MYCIN [6] and PROSPECTOR [7] have. Currently, there is little or no understanding for the nature, complexity, and modes of human interactions that may occur in a complex system like CK-LOG. The viability of using

*Expected to be compieted in Spring 1986.

†A trivial use of this feature occurred in the way we used 'risk-factor' in Section 5 to select a course of action from the various possibilities that were presented to the user by the system. Here the user made the judgment. In a future system, judgmental decisions like this may be automated within CK-LOG.

a CK-LOG based OPPLAN-CONSULTANT in the Navy will ultimately depend very much on the facilities that are made available for commander/machine communication. To understand the problems in this area, it is necessary to gain experience on the use of CK-LOG in this planning environment. Implementation of the example analyzed in this report will be the first step in this process.

Let me conclude this report with a historical note on MDS and the evolution of ideas that led to CK-LOG. MDS was first presented as a knowledge representation system in 1973 IJCAI at Stanford, CA. At that time, an initial implementation of MDS, called TEMPEST (a template establishment system) was available. What came to be called *frames*, after Minsky's report [18] were then called *templates* in MDS. I did not then have good ideas on how one might use the knowledge represented in MDS to state and solve problems. I thought mostly in terms of procedures written in a language which was then called the *designer* [3], but it was never implemented. The ideas proposed in MDS were first implemented into a working system by Sridharan [19]. This system was called AIMDS (Action Interpretation Meta Description System). It used the modelling system proposed in MDS with *procedural attachments* to solve problems.

The possibility of using the natural deduction system of Kanger in MDS was first explored in 1975. Because of several interruptions in my work, this idea was not pursued vigorously, until I started work on the Naval Operational Planning problem in 1982. This led to the results presented in this report.

7. ACKNOWLEDGMENT

I am thankful to the Navy Center for Applied Research in Artificial Intelligence for supporting this work since the summer of 1981.

8. REFERENCES

1. C.V. Srinivasan, *CK-LOG, A Calculus for Knowledge Processing in Logic*, 1984. Available in manuscript form from the author.

2. S. Kanger, "A Simplified Proof Method for Elementary Logic," in *Computer Programming and Formal Systems*, P. Braffort and D. Hirshberg, eds. (North-Holland Publishing Co., Amsterdam, 1963), pp. 87-94.

3. C.V. Srinivasan, "Architecture of Coherent Information System: A General Problem Solving System," *IJCAI-3*, 618-628 (1973). Republished in *IEEE Trans. on Computers*, **C-25** (4), 390-402 (1976).

4. C.V. Srinivasan, "The Meta Description System: A System to Generate Intelligent Information Systems, Part I, The Model Space," SOSAP-TR-20A, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1977.

5. C.V. Srinivasan, *Design for OPPLAN-CONSULTANT, An Expert System for Naval Operational Planning*, Final Research Report on work supported by ONR Grant 4330/NOO14-82-C-2126, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1983.

6. E.H. Shortliffe, *Computer-Based Medical Consultations: MYCIN* (American Elsevier, New York, 1976).

7. R.O. Duda et al. *Development of the Prospector Consultation System for Mineral Exploration*, Final Report to the Office of Resource Analysis, U.S. Geological Survey, Reston, VA, and to the

Mineral Resources Alternatives Program, The National Science Foundation, Washington, DC. Artificial Intelligence Center, SRI International, Menlo Park, CA, 1978.

8. G. Gentzen, "Investigations into Logical Deduction," *The Collected Papers of Gerhard Gentzen* (North-Holland Publishing Co., Amsterdam, 1935), pp. 68-131.

9. J. Irwing and C. V. Srinivasan, *Description of CASNET in MDS*, RUCBM-TR-51, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1975.

10. C.V. Srinivasan, *Programming over a Knowledge Base, the Basis for Automatic Programming*, SOSAP-TR-5, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1973.

11. R. Kowalski,. *Logic for Problem Solving* (North-Holland Publishing Co., Amsterdam and New York, 1979).

12. J.A. Robinson, "A Machine-oriented Logic Based on the Resolution Principle," *J. Assoc. Compt. Mach.*, **12**, 227-234 (1965).

13. D.G. Bobrow et al. "GUS, a Frame-Driven Dialog System," *Artificial Intelligence Journal*, **8** (2), 155-173 (1977).

14. I.P. Goldstein and R.B. Roberts, "Using Frames in Scheduling," in *AI-MIT*, **1**, 251-284 (1979).

15. J.G. Schmolze and R.J. Brachman, *Proceedings of the 1981 KL-One Workshop*, Bolt Beranek and Newman Inc., Report No. 4842, 1982. Prepared for Advanced Research Projects Agency.

16. M. Stefik, "An Examination of a Frame-structured Representation System," in *IJCAI-6*, 845-852 (1979)

17. J. McCarthy and P.J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," MI **4**, 463-502 (1969).

18. M. Minsky, "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, P.H. Winston, ed. (McGraw-Hill, New York, 1975) pp. 211-277.

19. N.S. Sridharan, *AIMDS User Manual-Version 2*, CBM-TR-89, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1978.

Appendix A
DEFINITION OF THE PLANNING KNOWLEDGE

A1. The COMMANDER and OPORDER Concepts

A.1.1. The Structure of COMMANDER Concept

The COMMANDER definition is shown in Fig. A1. According to this definition, for any COMMANDER, comndr, the phrase '(is-commander comndr),' will denote the mil-action commanded by the comndr. So also, (has-command-of comndr) is used to refer to the set of FORCEs that are under the command of the comndr. Also, as per the definition, each comndr reports-to another commander who is different from himself, may have subordinates who should also be different from himself. Each comndr issues OPORDERs, and himself may receive an OPORDER. He will be assigned to an OPORDER-TASK that is specified in the OPORDER received by him.

A COMMANDER may create an OPPLAN (Fig. A2 in Section 2) and may participate in the creation of other OPPLANs. The OPPLANs in whose creation a commander participates are those that are coordinated with him during their preparation and are so mentioned in the OPPREFACEs (Fig. 2.6 in Section 2). An OPPREFACE is a letter that announces the existence of an OPPLAN. A commander may receive OPPLANs for approval as indicated by the relation 'has-recd-for-approval' and may approve them ('has-approved'). Finally, a commander may be in the distribution list (isin-distn-list) of certain OPPREFACEs, especially those in which he is mentioned as having participated. The description of the comndr's command is specified by the (comnd-desn-for comndr). This is a COMND&SIGNAL specification.

A COMMANDER, comndr, may have STAFF-OFFICERs as his advisers. The 'advisers-of' relation has been declared as being *irreflexive* because a staff officer himself may be a commander: STAFF-OFFICER is a specialization of the COMMANDER concept. The comndr also has a second in command, who is denoted by (2nd-in-command-for comndr). The 'second in command' concept is specified by S-COMMANDER, (S for Second), which is also a specialization of the COMMANDER concept. A commander who is second in command for another COMMANDER may already be himself a commander of a MIL-ACTION. Thus, the S-COMMANDER concept will inherit the structure of COMMANDER concept. Also, it will also have the additional relational structure:

(is-2nd-in-command S-COMMANDER COMMANDER), *irreflexive.*

The irreflexive flag here specifies that a commander cannot be his own second in command.

A1.2. The Function of COMMANDER

Every, comndr, has three functions: the first is to KEEP operational every force that is under his command. In the logical statement shown in Fig. A1, the phrase (has-command-of comndr) refers to the set of forces under the command of the comndr. This statement may be read as follows: for every force that is in (has-command-of comndr), (KEEP (status-of force operational)) must be true. 'KEEP' here

Structure: (is-commander COMMANDER MIL-ACTION), 1
(has-command-of COMMANDER FORCES), 1
(reports-to COMMANDER COMMANDER), *Irreflexive*, 1
(subordinates-of COMMANDER COMMANDERS), *Irreflexive*
(advisers-of COMMANDER STAFF-OFFICERS), *Irreflexive*
(creates COMMANDER OPPLAN)
(participated-in COMMANDER OPPREFACES)
(issues COMMANDER OPORDERS)
(has-received COMMANDER OPORDER), 1
(has-been-assigned-to COMMANDER OPORDER-TASK), 1
(comnd-desn-for COMMANDER COMND&SIGNAL), 1
(isin-dist-list COMMANDER OPPREFACES)
(has-recd-for-approval COMMANDER OPPLANS)
(has-approved COMMANDER OPPLANS)
(2nd-in-command-for COMMANDER S-COMMANDER),
Irreflexive, 1

Function:
```
([(EVERY force (has-command-of comndr))
  (KEEP (status-of force operational))]
AND
[(EVERY opplan (has-recd-for-approval comndr))
  (OCCURS (CREATE (has-approved comndr opplan)))]
AND
[(EXISTS opplan OPPLAN)
 (EVERY oporder-task (has-been-assigned-to comndr))
 ([after
   (time-of
    (CREATE
     ([task-plan-for oporder-task opplan] AND
      [approved-by opplan (reports-to comndr)])))]
  [ACHIEVE (function opplan)])])
```

Behavior: None at the moment.

Analysis: None at the moment.

Design: None at the moment.

Fig. A1—The COMMANDER concept

refers to the processes that the commander may invoke to maintain the operational status of the force. The second function is to CREATE the approvals for the OPPLANs he has received for approval. The 'CREATE' operator here refers to the processes that the comndr may use to approve the OPPLANs (notice the difference here between the 'CREATE' operator and the 'creates' relation). '(OCCURS (CREATE ---))' asserts that such a CREATE process should occur; it may or may not result in the approval of the plans. In either case, '(OCCURS ---)' will be true as long as the process occurs. Thus, a COMMANDER will satisfy this part of his function whether a plan was approved by him or not, as long as he performed the approval process for the plan. This is to be contrasted with the ACHIEVE operation in the third function discussed below.

The third function is that, after the creation of the task-plan-for the assigned oporder-task and having it approved by his superior commander (the one he reports-to), he should proceed to ACHIEVE the *function* of this opplan. Notice that the CREATE expression here operates on a conjunctive logical expression. To make this conjunction true, it is quite possible that the CREATE processes involve repeated plan generation and approval cycles, as we see in Section A1. In general, the CREATE operating on a conjunction is not the same as the conjunction of CREATEs operating separately on each conjunct. Note also that this last function establishes a causal connection between the opplan creation and approval, and the achievement of the opplan *function*. It does not, however, specify when a commander should start creating the opplan for the oporder-task. This is specified by the *function* of OPORDER: he will start the planning process after he has received the OPORDER.

A1.3. Converse Relations

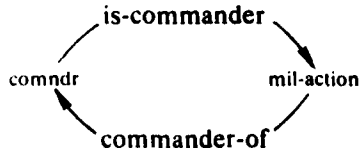
One should note the following in the interpretation of the above structure definition: a particular MIL-ACTION, say mil-action, with a particular COMMANDER, say comndr, should be such that

(commander-of mil-action comndr) and
(is-commander comndr mil-action)

are both either simultaneously true or simultaneously false. They constitute a *converse* pair of relations. The definitions:

(commander-of MIL-ACTION COMMANDER) and
(is-commander COMMANDER MIL-ACTION)

define this converse pair. They define the structure shown below:



This structure is declared to CK-LOG by asserting that (commander-of, is-commander) are *converses* of each other. Similarly, if (reports-to $comndr_1$ $comndr_2$) is true, then (subordinates-of $comndr_2$ $comndr_1$) should also be true. Thus, (reports-to, subordinates-of) are also converses. Notice that whereas a

commander reports to only one other commander, he may have several subordinates. Both these relations have been declared as being *irreflexive*, i.e., a commander may not report-to himself or be a subordinate of himself. There are other restrictions on the relations appearing in the structure of COMMANDER. These are presented in the next section.

A1.4. Restriction on COMMANDER Structure

COMNDR-RESN1: If x is a subordinate of comndr, then the comndr has-command-of every force that x has-command-of.

```
[(EVERY x (subordinates-of comndr))
(EVERY force (has-command-of x))
 (has-command-of comndr force)]
```

COMNDR-RESN2: For every oporder that a comndr issues, and is issued-to y, y is a subordinate-of comndr:

```
[(EVERY oporder (issues comndr))
 (EVERY y (issued-to oporder))
 (subordinates-of comndr y)]
```

COMNDR-RESN3: The OPORDER that a comndr has-received should be the one that ordered the MIL-ACTION that he commands:

```
[has-received comndr (ordered-by (is-commander comndr))]
```

Here, (is-commander comndr) refers to the mil-action that is commanded by the comndr.

COMNDR-RESN4: A comndr creates the OPPLAN for the MIL-ACTION that he commands:

```
(creates comndr (mil-plan-for (is-commander comndr))).
```

COMNDR-RESN5: If a comndr participated-in the preparation of an OPPLAN, then he should be in the distribution list of the OPPREFACE of that OPPLAN:

```
[(EVERY x OPPLAN)
 ((participated-in comndr x) IMPLIES
  (isin-distn-list comndr (distn-list-of (preface-of x))))]
```

COMNDR-RESN6: The OPORDER-TASK assigned to a comndr should be one of the tasks specified in the OPORDER received by him:

```
(has-been-assigned-to comndr (specifies-tasks (has-received comndr))
```

A1.5. The Structure of OPORDER

The concept of operation order, OPORDER, is defined in Fig. A2. The structure of OPORDER shown in Fig. A2 is intended to capture the operation order format given in NWP-11. In this structure, the phrase (orders OPORDER) will refer to the set of MIL-ACTIONs, ordered by the OPORDER. Here, (orders, ordered-by) are converses. Operation orders are issued-to commanders and issued-by commanders, where (issued-to, has-received) and (issued-by, issues) are converses. As the reader may have by now surmised, every binary relation in the system has an associated converse.

Structure: (orders OPORDER MIL-ACTIONs), 1
(is-produced-from OPORDER OPPLAN)
(opplans-for OPORDER OPPLANs), 1
(classification-of OPORDER SECURITY-CLN), 1
(heading-of OPORDER OPR-HEADING), 1
(issued-by OPORDER COMMANDER), *irreflexive*, 1
(issued-to OPORDER COMMANDERs), *irreflexive*, 1
(references-of OPORDER REFERENCEs), 1
(situation-of OPORDER MIL-SITUATION-DESN), 1
(objective-of OPORDER MIL-OBJECTIVE), 1
(purpose-of OPORDER MIL-OBJECTIVE), 1
(specifies-tasks OPORDER OPORDER-TASKs), 1
(logistics-of OPORDER OPR-LOGISTICS), 1
(comnd&signal-defns-of OPORDER COMND&SIGNALs), 1

Function: [(EVERY oporder-task (specifies-tasks oporder))
(EXISTS comndr (assigned-comndr-of oporder-task))
([after (*time-of* (has-received comndr oporder))]
[(after
(CREATE
([creates comndr (task-plan-for oporder-task)] AND
[approved-by (task-plan-for oporder)
(reports-to comndr)])))
(ACHIEVE (*function* (task-plan-for oporder-task)))])]

Behavior: (OCCURS (ACHIEVE (*function* oporder))).

Analysis: None at the moment.

Design: None at the moment.

Fig. A2—The concept of an operation order: OPORDER

The classification-of an OPORDER is its SECURITY-CLN (security classification) which will specify the level of the classification and associated instructions, if any. The heading-of an operation order is OPR-HEADING, with a structure that specifies the form of its contents. The meaning of other similar concepts that appear in the structure of OPORDER will become clear as they are defined later in this sec-

tion. Note that (specifies-tasks OPORDER) is intended to capture the parts of an OPORDER that describe the *task organization* and the *execution paragraph* of the OPORDER as specified in NWP-11. So also, the (comnd&signal-defns-of OPORDER) is intended to capture the *command and signal* specifications associated with an operational order.

An OPORDER will result in the generation of several OPPLANs, one for each OPORDER-TASK specified by the OPORDER. These are the plans denoted by (opplans-for OPORDER). Also, each OPORDER may itself be produced-from an OPPLAN.

The following observation on the structure of an OPORDER is noteworthy: every OPORDER has one principal objective and purpose. This is the objective and purpose specified by the OPPLAN from which the OPORDER itself was produced. In addition to this, it may also have several secondary objectives and purposes. These will appear in the OPORDER-TASKs specified by an OPORDER, as shown in the next subsection in the definitions of the components of an OPORDER. As in Fig. A1, each OPORDER-TASK is assigned to a COMMANDER who is the commander-of the MIL-ACTION spawned by the OPORDER-TASK. This COMMANDER is responsible for the generation of a task-plan-for the OPORDER-TASK that is assigned to him. This task plan will be a new OPPLAN generated by the COMMANDER for his task, and as shown in Fig. A1 will also be the mil-plan-for the MIL-ACTION specified by the OPORDER-TASK. Using this OPPLAN as his guide, the COMMANDER will create a new OPORDER, which he will issue to his subordinates. This is the recursive plan generation process that is captured by the structures of OPORDER (Fig. A2) OPORDER-TASK (Fig. A5) and OPPLAN (Fig. 2.5 in Section 2).

A1.6. Restrictions on OPORDER Structure

OPR-RESN1: This restriction is the same as COMNDR-RESN2 stated previously. We have here the following simpler statement:

[(EVERY comndr (issued-to oporder))
(subordinates-of (issued-by oporder) comndr)]

In CK-LOG, it is sufficient if a restriction is stated at one place. The interaction of the restriction with other concepts is automatically analyzed by CK-LOG. Usually, restrictions are stated at places where it is easy to define them.

OPR-RESN2: The purpose of an oporder should be the same as the purpose of the OPPLAN from which it was produced:

(purpose-of oporder (purpose-of (is-produced-from oporder)))

OPR-RESN3: The objective of an oporder should be the same as the objective of the OPPLAN from which it was produced:

(objective-of oporder (objective-of (is-produced-from oporder)))

OPR-RESN4: The (opplans-for oporder) are the task plans generated for the OPORDER-TASKs specified by the oporder:

[(EVERY x (specifies-tasks oporder))(opplans-for oporder (task-plan-for x))]

OPR-RESN5: An oporder is issued-to all the assigned COMMANDERs of its OPORDER-TASKs. The assigned COMMANDERs are also subordinates-of the one by whom the oporder was issued:

[(EVERY comndr (assigned-comndr-of (specifies-tasks oporder)))
([issued-to oporder comndr] AND
[subordinates-of (issued-by oporder) comndr])]

OPR-RESN6: An oporder is-produced-from an opplan. This opplan will be the task plan of an oporder-task. This oporder-task will itself be specified in another OPORDER, x, where x is the OPORDER that is received by the COMMANDER who issued oporder:

[is-produced-from oporder
(task-plan-for (specifies-tasks (has-received (issued-by oporder)))))]

A2. The OPORDER Components

A2.1. Structures of OPORDER Components

The structures of the OPORDER components are shown in Figs. A3 and A4. The structure of OPR-HEADING shown in Fig. A3 is self-explanatory. Note that (is-heading, heading-of) are converses and the abbreviation 'hq' has been used for 'headquarters.' The issuing headquarters of an OPR-HEADING is defined by the 'issuing-hq-of' relation. There are two possible choices for this: either a HEADQUARTERS, which will specify the name of the headquarters and its address, or HQ-CODE which is the code for the headquarters. For any given oporder, one is free to specify one or both of these. The date&time-of an OPR-HEADING is an instance of TIME which will specify the date and time at which the oporder was created.

Structure: (is-heading OPR-HEADING OPORDER), 1
(copy-no-of OPR-HEADING INTEGER), 1
(issuing-hq-of OPR-HEADING HEADQUARTERS), 1
(issuing-hq-of OPR-HEADING HQ-CODE), 1
(date&time-of OPR-HEADING TIME), 1
(reference-no-of OPR-HEADING REF-NUM), 1
(type-of OPR-HEADING OPR-TYPE), 1
(serial-no-of OPR-HEADING SERIAL-NUM), 1
(time-zone-of OPR-HEADING TIME-ZONE), 1

Function: None.

Analysis: None.

Design: None.

Fig. A3—The structure of an oporder heading: OPR-HEADING

HEADQUARTERS: (is-issuing-hq HEADQUARTERS OPORDERS), issuing-hq-of
(address-of HEADQUARTERS ADDRESS), is-address, 1
(address-of HEADQUARTERS FORCE-UNIT),
(code-of HEADQUARTERS HQ-CODE), is-code, 1

HQ-CODE: (is-code HQ-CODE HEADQUARTERS), 1
(is-issuing-hq HQ-CODE OPORDERS)

REFERENCE: (is-a-reference REFERENCE OPORDER), references-of, 1
(maps-of REFERENCE MAPS), is-a-map
(charts-of REFERENCE CHARTS), is-a-chart
(documents-of REFERENCE DOCUMENTS), is-a-document

MIL-SITUATION: (is-situation MIL-SITUATION OPORDER), situation-of, 1
(enemy-forces-in MIL-SITUATION FORCES), is-an-enemy-force
(enemy-objectives-in MIL-SITUATION MIL-OBJECTIVE)
(friendly-forces-in MIL-SITUATION FORCES),
(friendly-objectives-in MIL-SITUATION MIL-OBJECTIVE)
(mil-actions-in MIL-SITUATION MIL-ACTIONS)

MIL-SITUATION-DESN: Statement in DL that describes the properties of a MIL-SITUATION.

Fig. A4—Structures of OPORDER components

Hereafter, let us assume that if *function, analysis*, and/or *design* are not specified, then they are either vacuous, i.e., none exist, or will be introduced later as necessary. Also, I will exhibit the structures without explicitly displaying in each figure the title, *structure*, and where necessary, I will show the name of the converse relation after the relation definition, separated by a comma. These conventions have been followed in Fig. A4 which shows the definitions of HEADQUARTERS, HQ-CODE, REFERENCE, and MIL-SITUATION. Note the following in Fig. A4:

A HEADQUARTERS may be the issuing-hq-of more than one oporder. The address of a headquarters may be an ADDRESS or a FORCE-UNIT. The references cited by an OPORDER may contain MAPS, CHARTS, and DOCUMENTS. Definitions of these concepts are not presented here since they are not relevant to the discussion of the example presented in Section 4. The MIL-SITUATION is specified by giving the enemy forces and friendly forces in the region of the military actions associated with an OPORDER, the relevant enemy, and friendly objectives, and the ongoing MIL-ACTIONS. The concept MIL-SITUATION-DESN refers to descriptions of a mil-situation in the language DL.

The structures of the remaining components of an OPORDER are shown in Fig. A5. Every OPORDER-TASK has an objective and purpose. These need not be the same as the objective and purpose of the OPORDER in which the task is specified. As we have seen before, in Fig. A2, each OPORDER may have several tasks associated with it, each with its own objective and purpose. The objectives associated with the tasks are the secondary objectives of an OPORDER. The determination of the OPORDER-TASKS associated with an OPORDER is the principal part of the operational planning problem. Each OPORDER-TASK has a MIL-ACTION associated with it. This mil-action will be such that:

(ordered-by mil-action) = (specified-by oporder-task) = oporder.

OPORDER-TASK:

(specified-by OPORDER-TASK OPORDER), specifies-tasks, 1
(is-produced-from OPORDER-TASK OPPLAN-TASK), gives-rise-to
(assigned-forces OPORDER-TASK FORCES), assigned-to, 1
(objective-of OPORDER-TASK MIL-OBJECTIVE), 1
(purpose-of OPORDER-TASK MIL-OBJECTIVE), 1
(specifies OPORDER-TASK MIL-ACTION), specified-by, 1
(assigned-comndr-of OPORDER-TASK COMMANDER),
has-been-assigned-to
(task-plan-for OPORDER-TASK OPPLAN), oporder-task-of
(supports OPORDER-TASK OPORDER-TASKS), supported-by,
Irreflexive

COMND&SIGNAL:

(is-comnd&signal-defn COMND&SIGNAL OPORDER), 1
(is-comnd-desn COMND&SIGNAL COMMANDER), 1
comnd-desn-for
(signals-of COMND&SIGNAL SIGNALS), is-signal, 1
(recognition-of COMND&SIGNAL RECOGNITIONS), 1
(identification-of COMND&SIGNAL IDENTIFICATIONS), 1
(acknowledgements-of COMND&SIGNAL ACKNOWLEDGEMENTS)
(hq-of COMND&SIGNAL HEADQUARTERS), 1
(movements-of COMND&SIGNAL MOVEMENTS)
(code-words-of COMND&SIGNAL CODE-WORDS)
(code-names-of COMND&SIGNAL CODE-NAMES)
(liaison-of COMND&SIGNAL HEADQUARTERS), 1
(policy-of COMND&SIGNAL E-POLICY), 1

OPR-LOGISTICS: (is-logistics-for OPR-LOGISTICS OPORDER), etc.

Fig. A5—OPORDER-TASK, COMND&SIGNAL, and OPR-LOGISTICS

The deployment and engagement rules for this mil-action will be specified by its associated MIL-EXECUTION. The mil-action will also have a COMMANDER, specified by (commander-of mil-action), and the description of this COMMANDER will in turn specify his superior (the one he reports-to), his subordinates, and the description of his command through a COMND&SIGNAL specification. In this sense, an OPORDER-TASK not only specifies the MIL-ACTION, its assigned force units, and execution, but also the command hierarchy associated with the task and the command descriptions. The task-plan-for an OPORDER-TASK is the OPPLAN for the task. This would be created by the commander of the MIL-ACTION associated with the OPORDER-TASK.

Notice that, as shown in Fig. A2, an OPORDER may have several instances of COMND&SIGNAL associated with it *via* the comnd&signal-defns-of relation. Each COMND&SIGNAL specified by an OPORDER is uniquely associated with a COMMANDER of a MIL-ACTION that is spawned by one of its OPORDER-TASKs. This COMMANDER is denoted by the phrase (is-comnd-desn COMND&SIGNAL), 'desn' for 'description.' The OPORDER itself is denoted by (is-comnd&signal-defn COMND&SIGNAL). Each COMND&SIGNAL will

specify the signalling, recognition, identification, and acknowledgment instructions, electronic policy (E-POLICY), and the headquarters for the COMMANDER who is associated with it. The design of the COMND&SIGNAL specifications for the OPORDER-TASKs in a OPORDER is also a part of the operational planning problem.

Figure A5 does not show the full structure for OPR-LOGISTICS since the details of logistics planning are not discussed in this report. However, the methods of concept definition and problem solving discussed in this report may be used also to state and implement a logistics planning consultant. The *function, behavior, analysis*, and *design* aspects of the concepts presented in Fig. A5 are also not shown in the figure, but will be introduced as and when they are needed.

A2.2. Restriction on the Structure of OPORDER-TASK

Let me now state the restrictions on the objective and purpose of an oporder-task in relation to the objective and purpose of the oporder in which it occurs; namely, the oporder denoted by (specified-by oporder-task):

OPRTASK-RESN1: The oporder that specifies an oporder-task is denoted by (specified-by oporder-task). This oporder is-produced-from an opplan. Thus, the phrase '(is-produced-from (specified-by oporder-task))' in the expression below refers to this opplan. The condition below asserts that the oporder-task itself is-produced-from an opplan-task that this opplan specifies:

[is-produced-from oporder-task
(specifies-tasks (is-produced-from (specified-by oporder-task)))]

OPRTASK-RESN2: An oporder-task can support only other tasks of the oporder in which the oporder-task itself is specified:

[(EXISTS y OPORDER-TASK)
((supports oporder-task y) IMPLIES
(specified-by y (specified-by oporder-task)))]

OPRTASK-RESN3: The assigned-comndr-of an oporder-task is the COMMANDER of the specifies the oporder-task:

(assigned-comndr-of oporder-task (commander-of (specifies oporder-task)))

OPRTASK-RESN4: The objective-of an oporder-task should be the same as the objective-of the opplan-task from which it is produced:

[objective-of oporder-task (objective-of (produced-from oporder-task))]

OPRTASK-RESN5: The purpose-of an oporder-task is the same as the purpose of the opplan-task from which it is produced:

[purpose-of oporder-task (purpose-of (produced-from oporder-task))]

Similarly, the assigned-forces, specifies and supports relations of an oporder-task should also have the same values as the corresponding ones in the opplan-task. The statements of these restrictions will be similar to the restriction above. These restrictions in effect specify how one might generate a oporder task from an opplan task specification. The structures of OPPLAN and OPPLAN-TASK are discussed in Section A2. The next section presents the concepts FORCE and REGION.

A3. FORCE and REGION

A3.1. The Concept of FORCE

A partial specification of the *structure* of a FORCE adequate for our purposes in this report is shown in Fig. A6. The concepts, FLEET, TASK-GROUP, TASK-FORCE, and FORCE-UNIT are *specializations* of the concept of FORCE: they will inherit from FORCE its structure definition and share with FORCE some common characteristics of *function, behavior, analysis*, and *design*. But each may have its own specialized additional structures, functions, etc., as well. The elements in the structure of a FORCE that are shared by all its specializations are shown in Fig. A6.

Every FORCE is commanded-by a COMMANDER, has a location, a status, a type, and may belong-to (in the sense of being a part of) another FORCE and itself may consist-of other FORCEs. It may be assigned-to an OPORDER-TASK and thus will get associated with the objective and purpose of the OPORDER-TASK. A FORCE may defend (and be defended-by) a FORCE (which may be itself). It has a defensive-range, an offensive-range, and may strongly-defend a REGION surrounding itself. FORCEs belonging to a FORCE may be detached from it as a part of a task organization specified in an OPORDER. This is described by the concept, DETACHMENT. Similarly, there may be FORCEs that are attached to a force as a result of a task organization. This is described by ATTACHMENT.

Figure A7 shows some of the additional structures that the specializations of FORCE have. In this figure, in the structure of FLEET, notice that the relational forms:

(consists-of FLEET TASK-FORCEs)
(consists-of FLEET TASK-GROUPs),

are specializations of

(consists-of FORCE FORCEs),

because FLEET, TASK-GROUP, and TASK-FORCE are specializations of FORCE. This has the following significance: even though a FORCE may in general consist of any other FORCE, a FLEET is constrained to consist-of only of TASK-FORCEs and TASK-GROUPs. Notice that FLEETs have the 'flag-ship' property and not all FORCEs have this property.

A TASK-GROUP may consist-of only TASK-FORCEs. Forces in a TASK-GROUP will have a common purpose and objective associated with them. This purpose and objective will be specified by the OPORDER-TASK to which the TASK-GROUP is assigned. Both TASK-GROUP and TASK-FORCE have the 'flag-ship' property.

FORCE: (*specializations-of* FORCE
(FLEET TASK-GROUP TASK-FORCE FORCE-UNIT))
(location-of FORCE REGION), is-location, 1
(commanded-by FORCE COMMANDER), has-command-of, 1
(type-of FORCE FORCE-TYPE), is-type, 1
(status-of FORCE FORCE-STATUS), is-status, 1
(belongs-to FORCE FORCE), consists-of, *Irreflexive*, 1
(consists-of FORCE FORCEs), *Irreflexive*, 1
(assigned-to FORCE OPORDER-TASK), assigned-forces, 1
(defends FORCE FORCEs), defended-by
(defended-by FORCE FORCEs)
(enemy-of FORCE FORCEs), enemy-of, *irreflexive*
(defensive-range-of FORCE RANGE),
(strongly-defends FORCE REGION), strongly-defended-by
(offensive-range-of FORCE RANGE),
(is-issuing-unit-for FORCE DETACHMENTs), detached-from
(detached-by FORCE DETACHMENT), detached-forces-in
(attachments-to FORCE ATTACHMENTs), is-attachment
(attached-by FORCE ATTACHMENT), attached-forces-in,
etc.

DETACHMENT: (detached-from DETACHMENT FORCE), 1
(detached-forces-in DETACHMENT FORCEs), detached-by, 1
(*time-of* DETACHMENT TIME), 1
(location-of DETACHMENT REGION), 1

ATTACHMENT: (is-attachment ATTACHMENT FORCE), 1
(attached-forces-in ATTACHMENT FORCEs), 1
(*time-of* ATTACHMENT TIME), 1
(location-of ATTACHMENT REGION), 1

Fig. A6—Structures of FORCE, DETACHMENT, and ATTACHMENT

FLEET: (consists-of FLEET TASK-GROUPs)
(consists-of FLEET TASK-FORCEs)
(flag-ship-of FLEET FORCE-UNIT)

TASK-GROUP: (belongs-to TASK-GROUP FLEET)
(consists-of TASK-GROUP TASK-FORCEs)
(flag-ship-of TASK-GROUP FORCE-UNIT)

TASK-FORCE: (belongs-to TASK-FORCE TASK-GROUP)
(belongs-to TASK-FORCE FLEET)
(consists-of TASK-FORCE FORCE-UNITs)
(flag-ship-of TASK-FORCE FORCE-UNIT)

FORCE-UNIT: (is-flag-ship FORCE-UNIT FORCE)
(belongs-to FORCE-UNIT FORCE)
(*specializations-of* FORCE-UNIT
(CV CG CLG DD FF DDG FFG SS SSN SSBN, etc))

Fig. A7—Structures of specializations of FORCE

FORCE-UNITs are further specialized into CV's (carriers), CG's (Guided Missile Cruisers), CLG's (Guided Missile Light Cruisers), DD's (Destroyers), FF's (Frigates), DDG's (Guided Missile Destroyers), FFG's (Guided Missile Frigates), SS (Submarine), SSN (Nuclear Attack Submarine), SSBN (Nuclear Ballistic Missile Submarine), etc. To evaluate and compare the capabilities of force units in relation to a given objective and known enemy forces, one needs much more details than what the structures given above provide. I will not discuss the details in this report. The next section presents some of the properties of the concept of REGION.

A3.2. The Concept of REGION

We have seen quite a few REGIONs so far, but I have not discussed how this concept is represented in OPPLAN-CONSULTANT. REGION is a specialization of OBJECT. In fact, ACTION, SET, and STRING are the only concepts that are not specializations of OBJECT. I will choose a highly simplified representation for REGION. This is shown in Fig. A8.

Regions are made up of BASIC-REGIONs. Each BASIC-REGION is a RECTANGLE or a CUBE specified by the coordinates (longitude and lattitude) of its center and the sizes of its sides. Odd-shaped REGIONs are viewed as made of several rectangular basic regions, overlapping if necessary. The inclusion relation is, 'is-within,' (is-within x y) and (expansion-of y x) are true if y is with in x. All the common relations between regions, such as 'adjacent-to,' 'outside,' 'overlaps-with,' 'contains,' etc. are made available. The distance between two regions is a tuple, '(distance x y z),' where x and y are regions, and z is distance in some unit, say MILEs, where MILE is a specialization of INTEGER. The distance is always between the designated centers of the regions. The center of a BASIC-REGION is always the center of the RECTANGLE or CUBE, but for other regions, the center can be any designated LOCATION in the region. Notice below that the 'center-of' relation applies to all regions, but 'length-of,' 'width-of,' and 'height-of' applies only to BASIC-REGIONs, and 'thickness-of' applies only to CUBEs (one may

REGION: (distance REGION REGION MILE)
(center-of REGION LOCATION), is-center, 1
(is-within REGION REGIONS), extension-of, *Irreflexive*
(contains REGION OBJECTS), isin
(height-range-of REGION MILE-PAIR), is-height-range, 1
(depth-range-of REGION MILE-PAIR), 1
(adjacent-to REGION REGIONS),
adjacent-to, *Irreflexive, Symmetric*
(has-access-to REGION FORCES)
(controlled-by REGION FORCE), controls
(is-in-offensive-range-of REGION FORCES)
(is-in-defensive-range-of REGION FORCES)
(is-strongly-defended-by REGION FORCES)
(*specializations-of* REGION
(WATER-REGION LAND-REGION AIR-REGION
BASIC-REGION LOCATION))
(*specializations-of* LAND-REGION
(CONTINENT ISLAND COUNTRY, etc.))
(*specializations-of* WATER-REGION
(OCEAN LAKE RIVER, etc.))
(*specializations-of* BASIC-REGION (CUBE RECTANGLE))
(width-of BASIC-REGION MILE), is-width, 1
(length-of BASIC-REGION MILE), is-length, 1
(thickness-of CUBE MILE), 1
(is-location LOCATION OBJECTS), location-of
(lattitude-of LOCATION NUMBER), 1
(longitude-of LOCATION NUMBER), 1
(depth-of LOCATION MILE), 1
(height-of LOCATION MILE), 1
(*is-a-specialization* CUBE (AIR-REGION WATER-REGION))
(*is-a-specialization* RECTANGLE
(LAND-REGION WATER-REGION)),
etc.

Fig. A8—Structure of REGION and its specializations

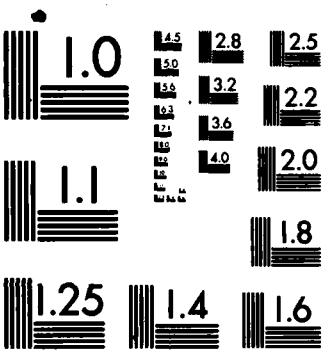
assume the thickness to be in the vertical dimension, length along the north-south dimension and width along the east-west dimension). LOCATION is a point in a REGION with a longitude, latitude, and sometimes also height and/or depth. The property (location-of region) will refer to its center.

Of interest to the operational planning domain are the relations 'controlled-by,' 'is-in-offensive-range,' 'is-in-defensive-range,' 'has-access-to,' etc. Figure A8 shows several *specialization-of* of REGION, such as WATER-REGION, LAND-REGION, AIR-REGION, BASIC-REGION, RECTANGLE, CUBE, etc. A RECTANGLEs and CUBEs may also be specializations of WATER-REGION, LAND-REGION, and AIR-REGION. There are other relations that assign attributes to REGIONs concerning their terrain, weather, etc., and also several other specializations of regions. It is not necessary to consider all the details here.

AD-A160 999 THE USE OF CK-LOG FORMALISM FOR KNOWLEDGE REPRESENTATION AND PROBLEM SOLV (U) RUTGERS - THE STATE UNIV NEW BRUNSWICK NJ LAB FOR COMPUTER SC

2/2

UNCLASSIFIED C V SRINIVASAN 30 SEP 85 NRL-8902 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Appendix B
INFERENCE RULES FOR QUANTIFIED EXPRESSIONS AND OPERATOR EXPRESSIONS

B1. Calculus of Sequents for Quantified Expressions

Table B1 specifies rules for quantified expressions. The first rule in this table is the *existential instantiation* rule, called [E →], for short. This replaces an existentially quantifier expression of the form, ((EXISTS x **range**) **exp**), by the two expressions '(eIc in **range**), (**subst** eIc x **exp**).' Here eIc is a *new variable* not used previously in the deduction tree in which the bottom sequent of the rule appears as a problem. This variable will eventually be bound to a constant if and when the proof is completed, i.e., when all the terminal sequents become axioms. It is the creation of this constant that gives this rule its name, 'existential instantiation.' This constant is required to be a member of the set specified by the **range**. Also, this constant is a function (is dependent) on all the variables uGx and uIc that might already appear in **exp**; these variables might have been introduced into **exp** by previous applications of the [→ U] and [U →] rules in the deduction tree. This dependency is indicated by the binding condition in the rule:

'eIc = (FEI uGx's & uIc's in **exp**),'

where 'FEI' is the name of a dummy function. It is used just to express the fact that eIc is dependent on its arguments. FEI is called the *Skolem function.* **Subst** is a function which is executed at the time of application of this rule. It substitutes eIc for every occurrence of the variable, x, in **exp**. The effect of this rule is to eliminate the 'EXISTS' quantifier from the sequent.

The *existential generalization* rule, [→ E], is quite similar to the existential instantiation rule. Here also the quantifier expression is replaced by two new expressions similar to the ones used above. But, in this case, instead of substituting the quantifier variable by the new variable eIc, it is substituted by the new variable eGx. eGx is called the generalization variable. The value of this variable is also required to be in the set specified by the **range** given in the quantification, and is dependent on the variables uGx and uIc that already appear in the expression, as specified by the binding condition in the rule. Here, the binding condition uses the *Skolem function* FEG.

The ϵ-expressions appearing in the rules in Table B1 are given special interpretation during variable binding process. They are used to restrict the possible bindings for the variables. In cases where the **range** is a set of constants, an attempt will be made to use the constants directly to bind variables in the expressions. If **range** is not a set of constants, then they can be either a union of concepts, X, or they may be expressions of the form '(r x)' where (r x) = {y | (r x y)}, or combinations of these.

The difference between eIc and eGx is the following: whereas to bind eIc one may create a new constant, *eGx is required to be bound only to a constant that has been already created in the deduction tree*, i.e., a constant that was created in the deduction tree before the application of this rule. Similarly, we

Table B1—Quantifier Elimination Rules

Rule Name	Inference
[E →]	*Existential Instantiation*
	.., (**elc in range**), (**subst elc x exp**), ... → .... ; Binding: elc = (FEI uGx's & ulc's in **exp**).
	.., ((EXISTS x **range**) **exp**), ... → .... ;
[→ E]	*Existential Generalization*
	.... → ..., (**eGx in range**), .. ; .... → ..., (**subst eGx x exp**), .. ; Binding: eGx = (FEG ulc's & uGx's in **exp**).
	.... → ..., ((EXISTS x **range**) **exp**), .. ;
[→ U]	*Universal Instantiation*
	...., (**ulc in range**) → ..., (**subst ulc x exp**), .. ;
	...., → ..., ((EVERY x **range**) **exp**), .. ;
[→ U]	*Universal Generalization*
	..., (**subst uGx x exp**), .. → .... ; ..., .. → (**uGx in range**), ... ;
	..., ((EVERY x **range**) **exp**), .. → .... ;

also have the *universal instantiation* and *universal generalization* rules called [→ U] and [U →], respectively. The universal instantiation and generalization variables are independent variables; i.e., they are not functionally dependent on any of the variables that might already appear in **exp**. Notice that [U →] and [→ E] split the problem sequent into two new sequents. Also, note that each application of a quantifier elimination rule eliminates only the outermost quantifier in a quantified expression.

Repeated application of these rules will eliminate all the quantified expressions from the problem sequents and replace them by the appropriate propositional expressions. A general rule that governs the application of the rules given in Tables 3.1 (in Section 3) and B1 is that:

> quantifier elimination rules may be applied to a problem sequent only after applying to the problem sequent all applicable propositional rules.*

Thus, after each application of a quantifier elimination rule, one should apply to the resultant sequent(s) all the applicable propositional rules before applying the next quantifier elimination rule.

Every time a quantifier elimination rule is applied, the inference engine will make a note of the position in the deduction tree at which the new variable generated by the rule was created. This will result in a partial ordering of the variables themselves under the successor (predecessor) relationship:

*There are several exceptions to this rule not enumerated here.

> A variable, x, is a successor to another variable, y, if x was created after y in the deduction tree. In this case, y is the predecessor of x.

During the proof process, values will be assigned to the generalization variables, where the values will be the constants created through the instantiations. The assignment of such values to variables should always obey the following important condition:

> A variable, x, may be assigned the value of another variable, y, only if y is a predecessor of x (i.e., y was created before x in the deduction tree). If all the variables created at the lowest level of a deduction tree are generalization variables, then one or more of them may be used as instantiation variables.

After assigning values to the generalization variables, the problem sequents in the leaf nodes of the deduction tree are given the axiom test (i.e., tested to see whether they are axioms or not). If all the leaf node sequents are not axioms, then the problem at the root is valid. If the leaf node sequents are not axioms, then the deduction is continued. It is quite possible that the deduction tree for a problem with quantified expression may never terminate because, as we see below, the generalization variables may be repeatedly regeneralized, and so also, the instantiation variables may be repeatedly reinstantiated. But if the problem sequent is valid, then it is guaranteed to terminate. The process of assigning values to variables and testing for axioms is discussed in Ref. 1. Some examples of these processes are presented in Section 3.

B2. Calculus of Sequents for Operator Expressions

The inference rules are presented below following the convention established in Section 3. Bold items in the rules refer to *syntactic* functions or patterns. Thus, for example in the first rule, namely, the *achieve-elimination* rule, **'value-of'** refers to the syntactic function which returns the value of its argument in the current world state. If the value is undefined in the current world state, then it will return the **value-of** expression itself as its value. **'Term'** is a syntactic pattern that stands for a function term (introduced in Section 3), **sexp** denotes a simple logical expression with or without operators but with no time parameters, and **(tm-term exp)** denotes a timed-expression where **tm-term** is the term that specifies the time. The pattern, **exp**, is used to denote an expression which may or may not be a timed one. Finally, **atm-sexp** denotes an *atomic simple expression*, i.e., one of the form **(relation-nm term-1 term-2)** or **(tuple-name term-1 ... term-n)** with or without the negation operator NOT, and **atm-exp** denotes a timed or untimed atomic expression. Each rule is followed by a brief comment.

In all the inference rules, the top statement may be viewed as defining the meaning of the bottom one. The discussion here will only provide a preliminary introduction to the rules. The reader will get a better understanding for the rules in the ensuing subsection where the use of these rules in the system's problem solving process is illustrated. Let me begin with the achieve elimination rule.

This is called [ACH-EL], for short. The condition, T, indicates that this rule is unconditionally applicable. The absence of '⟶' symbol in the rule indicates that the same rule applies both to the left and the right side of a sequent: the rule may be applied uniformly without regard to where the ACHIEVE statement occurs in a sequent, even if it occurs embedded in another larger expression. In this sense,

the rule is said to be context independent. Informally, one may think of the above rule as stating that for any **term**, (ACHIEVE **term**) always means, CREATE, whatever the **value-of the term** denotes in the current world state. Most of the rules presented below are context independent rules. The rules in the next table are used to eliminate ISTRUE, ISFALSE, and ISUNKNOWN operators. Their interpretation is quite straightforward.

Table B2—Achieve Elimination Rule

Name	Condition	Inference
[ACH-EL]	T	(CREATE (**value-of term**))
		(ACHIEVE **term**)
[ACH-RED]	T	(ACHIEVE **term**)
		(ACHIEVE (ACHIEVE **term**))

Table B3—ISTRUE, ISFALSE, and ISUNKNOWN Elimination

Name	Condition	Inference
[IST-EL1]	T	(**value-of term**)
		(ISTRUE **term**)
[ISF-EL]	T	[NOT (**value-of term**)]
		(ISFALSE **term**)
[IS?-EL]	(ISUNKNOWN (**value-of term**))	T
		(ISUNKNOWN **term**)
[IST-CR]	T	(ISTRUE **exp**)
		(ISTRUE (CREATE **exp**))
[IST-DES]	T	(ISTRUE (NOT **exp**))
		(ISTRUE (DESTROY **exp**))
And similar rules for [ISF-CR], [ISF-DES], etc.		

Notice that the rule [IS?-EL] is applied only if (ISUNKNOWN **term**) is true. If it is true, then it is replaced by T, otherwise it is left unchanged. The rules in the next table give us a way of eliminating (*time-of* **event**) or (*interval-of* **event**) expressions from a sequent. These expressions are called *event-time* expressions. There are two cases corresponding to whether the **event** is true in the current world state or not. The syntactic variable, **event-rel**, stands for '*time-of*' or '*interval-of*,' and the variable **time-rel**, stands for 'after,' 'before,' 'during,' 'between' or 'at.' Note that 'during' and 'between' relations will appear only with the '*interval-of*' expressions and so also, 'at' will appear only with '*time-of*' expressions. If an expression contains variables which have no bindings as yet specified, then the value of the expression in the world state will be ? (unknown), and in this case the ISTRUE function for that term will return F.

Table B4—Event-time Elimination Rules

Name	Condition	Inference	Bindings
[EVTM-EL1]	(ISTRUE **event**)	[(**time-rel** ei-t) **exp**]	(ei-t = (**value-of** (**event-rel event**)))
		[(**time-rel** (**event-rel event**)) **exp**]	
[EVTM-EL2]	(ISTRUE **event**)	[(**time-rel** ei-t) (**operator exp**)]	(ei-t = (**value-of** (**event-rel event**)))
		[**operator** ((**time-rel** (**event-rel event**)) **exp**)]	
[EVTM-EL3]	(NOT (ISTRUE **event**))	([ei-t **event**) IMPLIES [(**time-rel** ei-t) **exp**])	(ei-t = (**value-of** (**event-rel event**)))
		((**time-rel** (**event-rel event**)) **exp**)	
[EVTM-EL4]	(NOT (ISTRUE **event**))	([ei-t **event**) IMPLIES [**operator** ((**time-rel** ei-t) **exp**)])	- (ei-t = (**value-of** (**event-rel event**)))
		(**operator** ((**time-rel** (**event-rel event**)) **exp**))	

The inference rule, [EVTM-EL1], specifies that if the **event** is true in the current world state, then the event expression may be replaced by the new variable, ei-t, the value of which is the time or interval of the **event**. [EVTM-EL2] indicates the role of **operator** under these conditions. **Operator** here is any operator other than ISTRUE, ISFALSE, ISUNKNOWN, and OCCURS. The **operator** does not operate on the **event**, but only on the **exp**. If the **event** is not true in the current world state, then the two cases are indicated by [EVTM-EL3] and [EVTM-EL4]: here the **event** implies the **exp** with the indicated time relationships.

Once the **event-rel** expressions are eliminated from a sequent using the [EVTM-ELi] rules, the sequent will contain only **time-rel** expressions. These are reduced using the time reduction rules, TMR-x, shown in Table B5. These rules show how complex **time-rel** expressions may be reduced to combinations of simpler ones. All of them are independent of the context of their appearance in a sequent. The first four are, [TMR-U] for universally quantified expressions, [TMR-E] for existentially quantified expressions, [TMR-AND] for conjunctions, and [TMR-OR] for disjunctions. As mentioned before, syntactic variable, **tm-term**, stands for a term that specifies time.

The remaining rules shown in Table B5 reduce embedded timed expressions. Thus, the rule [TMR-TM] gives the condition for replacing (x (y **exp**)) by (x **exp**); namely, that (x = y), where x and y are either variables or constants (i.e., known time instances). If they are constants, then the rule is applicable only if they are equal, and if x or y is an unbound variable, then the binding condition specifies that they should be equal. Similarly, [TMR-XN] specifies the rule for reducing arbitrary embedded time expressions. Here, **tmxn** is the function that modifies the expression to account for the intersection of the two time terms in the expression. Thus, for example, (**tmxn** ((after x) (during (y z) exp), will be ((during (y z) exp) if y is after x, ((during (x z) exp) if y is not after x but z is after x, and NIL if x is after z. **tmxn** is defined for all possible combinations of time terms. Note that in general (**tmxn** (term1 (term2 exp)) is not the same as ((term1 exp) AND (term2 exp)). In the examples we consider in this report, **tmxn** is applied only to terms containing known time instances. I will not

Table B5—Time Reduction Rules

Name	Condition	Inference	Bindings
[TMR-U]	T	((EVERY x (tm-term range))(tm-term exp))	None
		(tm-term ((EVERY x range) exp)))	
[TMR-E]	T	((EXISTS x (tm-term range))(tm-term exp))	None
		(tm-term ((EXISTS x range) exp)))	
[TMR-N]	T	(NOT (tm-term exp))	None
		(tm-term (NOT exp))	
[TMR-AND]	T	((tm-term exp-1) AND (tm-term exp-2))	None
		(tm-term (exp-1 AND exp-2))	
[TMR-OR]	T	((tm-term exp-1) OR (tm-term exp-2))	None
		(tm-term (exp-1 OR exp-2))	
[TMR-TM]	(x = y)	(x exp)	(x = y)
		(x (y exp))	
[TMR-XN]	T	(tmxn (tm-term-1 (tm-term-2 exp)))	None
		(tm-term-1 (tm-term-2 exp))	

therefore present here the inference rules for **tmxn**. *The rules for* CREATE are presented next in Tables B6a and B6b.

Note that the inference rules for CREATE may be applied to a CREATE expression, even if it occurs embedded inside another larger expression. All top level CREATE expressions (those that do not appear embedded in other expressions) are retained in a sequent. The rules are applied to copies of such expressions, which are introduced into the sequents before the application of the rules. The system keeps a record of these expressions in a sequent which have been thus copied and expanded. This same convention also applies to DESTROY and KEEP expressions.

There are fourteen inference rules for CREATE, two elimination rules, [CREL-i], one reduction rule for timed expressions, [CRR-TM], nine for logical expressions (five for propositional combinations, and four for quantifier expressions), and two for reducing operator combinations with CREATE as the first operator. If an **exp** is true in the current world state, then it has already been created. In this case, as per [CREL-1] one may simply replace the CREATE expression by **(true-part-of exp)*** in a sequent. [CREL-2] says that if the argument of CREATE is a positive atomic expression, **patm-exp**, and it is not true in the world state, then it is to be replaced by the action that is needed to create it. Here, **patm-exp** can also be the name of an object or action. If it is an object, then it will be replaced by the action needed to create the object and its *prompt* relations.

*The part of the **exp** that evaluates to T in the world state, and which caused the **exp** itself to have the truth value T. Thus, for example, the true part of (x OR y) when x is false and y is true is y; if both are true, then it is (x OR y), if x is true and y is false, then it is x, otherwise it is NIL.

Table B6a—Reduction Rules for CREATE Expressions

Name	Condition	Inference
[CREL-1]	(ISTRUE **exp**)	(**true-part-of exp**)
		(CREATE **exp**)
[CREL-2]	(NOT (ISTRUE **patm-exp**))	(ASSERT (**create-action-of patm-exp**))
		(CREATE **patm-exp**)
[CRR-TM]	T	(CREATE (**tm-term exp**))
		(**tm-term** (CREATE **exp**))
[CRR-DNF]	T	(CREATE (**dnf conjunction**))
		(CREATE **conjunction**)
[CRR-NOT]	T	(DESTROY **exp**)
		(CREATE (NOT **exp**))
[CRR-IMP]	T	([DESTROY **exp-1**)] OR [CREATE **exp-2**])
		(CREATE (**exp-1** IMPLIES **exp-2**))
[CRR-AND]	T	(ASSERT (.., [CREATE **exp**], ...))
		(CREATE (..AND **exp** AND...))
[CRR-OR]	T	(..OR [CREATE **exp**] OR...)
		(CREATE (..OR **exp** OR...))
[CRR-OP1]	T	(**operator-a exp**)
		(CREATE (**operator-a exp**))
[CRR-OP2]	T	(**tm-term** (**operator-a exp**))
		(CREATE (**tm-term** (**operator-a exp**)))
[CRR-ASRT]	T	(ASSERT ((CREATE **exp**), ..., (CREATE **exp**)))
		(CREATE (ASSERT (**exp**, ..., **exp**)))

Table B6b—Reduction Rules for Quantified CREATE Expressions

Name	Condition	Inference
[CRR-U →]	T	(.., [(DESTROY (ug-x in range)) OR (CREATE (subst ug-x x exp))], ...) → ;
		(.., (CREATE ((EVERY x range) exp)), ...) → ;
[→ CRR-U]	T	→ (.., [(DESTROY (ui-c in range)) OR (CREATE (subst ui-c x exp))], ...) ;
		→ (.., (CREATE ((EVERY x range) exp)), ...) ;
[CRR-E →]	T	(.., (ASSERT ((CREATE (ei-c in range)), (CREATE (subst ei-c x exp)))), ...) → ;
		(.., (CREATE ((EXISTS x range) exp)), ...) → ;
[→ CRR-E]	T	→ (.., (ASSERT ((CREATE (eg-x in range)), (CREATE (subst eg-x x exp)))), ...) ;
		→ (.., (CREATE ((EXISTS x range) exp)), ...) ;

The rule, [CRR-TM], says that time specifications may be moved into a CREATE expression, from outside. The rules [CRR-DNF], [CRR-U], [CRR-E], [CRR-AND], [CRR-IMP], [CRR-OR], and [CRR-NOT] specify methods for decomposing the creation of complex logical expressions into combinations of creation of simpler ones. The pattern, '(..AND exp AND...)' in [CRR-AND] rule refers to a conjunction of expressions, where exp may be followed on either side by zero or more other conjuncts (thus, the pattern can be simply 'exp' itself). Each expression, exp, in the conjunction is replaced by '(CREATE exp),' inside the scope of ASSERT, and the AND connectives are replaced within the ASSERT by commas.

[CRR-AND] requires that the conjuncts in such a conjunction should all be ASSERTed jointly. This is indicated in the rule by the occurrence of the CREATE expressions nested within the outer ASSERT. Each inner CREATE expression, in this nesting, will be reduced first either to ASSERT expressions or simply to atomic expressions using the appropriate logical rules and elimination rules. This will result in either a single assertion containing one or more atomic expressions, or a disjunction of such assertions. In each such assertion, the atomic expressions are effectively asserted *in parallel* into the world state. The inner CREATE expressions will get reduced as follows:

If the argument of an inner CREATE expression is a quantified expression, then the quantifier rules of Table B6b will be used until the argument is reduced to a conjunction, disjunction, a negation, or an implication. If the argument is a negation, then it will be changed to a DESTROY expression, using [CRR-NOT]. If the argument is an implication, then [CRR-IMP] will be used. If it is a disjunction, then [CRR-OR] will be used. If it is a conjunction, then it will be first put in *disjunctive normal form* (dnf), i.e., into a disjunction of conjunctions, as for example in:

$$((x \text{ OR } y) \text{ AND } z) = ((x \text{ AND } z) \text{ OR } (y \text{ AND } z))^*.$$

*The dnf of (x AND y) = (x AND y).

This is accomplished by using the [CRR-DNF] rule. In general, a disjunctive normal form expression is a disjunction of conjunctions. Thus, [CRR-OR] rule may be used after [CRR-DNF] to separate out the disjunctions. This process is iterated until one gets CREATE expressions consisting only of conjunctions of atomic expressions. In this process, it may often be necessary to employ the [CRR-OPi], [ASRT-OP], [ASRT-ASRT], and [ASRT-OR] rules in Table B9b to get rid of certain nested CREATE (and ASSERT) expressions. *[CRR-OPi] specifies that the creation of an* **operator-a** expression is the same as the **operator-a** expression itself, where **operator-a** is the same as **operator**, but excluding ASSERT. The [ASRT-ASRT] specifies that inner ASSERTs in nested ASSERT expressions may be reduced to the arguments of the ASSERT expressions.

Table B6b shows the reduction rules for quantified expressions. The pattern (.., quantifier-exp, ...) indicates a form that might appear inside an ASSERT. I have not shown the ASSERT itself in Table B6b, because I wanted to indicate that the same rules apply also to the case where '(CREATE quantified-exp)' appears alone in the sequent. The expressions '(.. in **range**)' in this table are the *binding conditions.* These conditions indicate the ranges of the variables that are newly generated by the application of these rules. Notice that the top expression in the inference of [CRR-U →] may be viewed as the result of application of ([U →], [CRR-IMP], [OR →]),* in the left to right order, to the expression at the bottom of the inference. The only difference is that [CRR-U →] allows one to apply [U →] to the universally quantified expressions that occur inside the scope of a CREATE. The other rules in Table B6b may be similarly interpreted. The **subst** function used in these rules is the same function introduced first in Section 3: (**subst** x y exp) substitutes x for all occurrences of y in exp. This is evaluated at the time of rule application. These are the *quantifier elimination* rules for CREATE expressions.

The variables ug-x and eg-x in the rules [CRR-U →] and [→ CRR-E] are the *generalization variables* first mentioned in Section 3. These variables are used to represent *arbitrarily selected* items from their respective **ranges**. As mentioned in Section 3, these variables *must be new variables, not previously used in the inference process in which they occur.* Similarly, ei-c and ui-c are *existential* and *universal instantiation* variables. These denote distinct constants that are *newly instantiated* in the inference process.

The inference rules in Tables B6a and B6b together specify the means for eliminating the CREATE operators from a sequent. The creation of a complex logical expression is decomposed to the creation of its simpler components, and the creation of the simplest unit is reduced to the actions that are needed to create it. The important point to note here is that, in general '([CREATE x] AND [CREATE y])' is not the same as '(CREATE (x AND y))' for expressions x and y. This is because, even though [CREATE x] and [CREATE y] may both be true, taken separately, their joint creation may not be always true. Thus, for example, one may be able to buy a car and buy a house, but may not have the necessary resources to buy both.

Similar rules for the DESTROY operation are shown in Tables B7a and B7b. The rules here are the dual of the rules in CREATE. For an arbitrary **exp** (that is not a positive atomic expression), if the **exp** is true in the current world state, then the **true-part-of** the **exp** is to be destroyed. For a **patm-exp**,

*Notice that [U →] and [OR →] were introduced in Section 3.

Table B7a—Reduction Rules for DESTROY Expressions

Name	Condition	Inference
[DES-EL1]	(ISFALSE **exp**)	(NOT **exp**)
		(DESTROY **exp**)
[DES-EL2]	(ISTRUE **patm-exp**))	(ASSERT (**destroy-action-of patm-exp**))
		(DESTROY **patm-exp**)
[DES-RED]	(ISTRUE **exp**)	(DESTROY (**true-part-of exp**))
		(DESTROY **exp**)
[DES-TRM]	T	(DESTROY (**value-of term**))
		(DESTROY **term**)
[DES-TM]	T	(DESTROY (**tm-term exp**))
		(**tm-term** (DESTROY **exp**))
[DES-CNF]	T	(DESTROY (**cnf disjunction**))
		(DESTROY **disjunction**)
[DES-NOT]	T	(CREATE **exp**)
		(DESTROY (NOT **exp**))
[DES-IMP]	T	(ASSERT ([CREATE **exp-1**], [DESTROY **exp-2**]))
		(DESTROY (**exp-1** IMPLIES **exp2**))
[DES-AND]	T	([DESTROY **exp-1**] OR [DESTROY **exp-2**]))
		(DESTROY (**exp-1** AND **exp-2**))
[DES-OR]	T	(ASSERT (.., (DESTROY **exp**), ...))
		(DESTROY (..OR **exp** OR...))
[DES-ASRT]	T	((DESTROY **exp**) AND ... AND (DESTROY **exp**))
		(DESTROY (ASSERT (**exp**, ..., **exp**)))

Table B7b—Reduction Rules for Quantified DESTROY Expressions

Name	Condition	Inference
[DES-U →]	T	(ASSERT (.., ([CREATE (ei-c in **range**)], [DESTROY [**subst** ei-c x **exp**])), ...) → ;
		(.., (DESTROY ((EVERY x **range**) **exp**)), ...) → ;
[→ DES-U]	T	→ (.., (ASSERT ([DESTROY (**subst** eg-x x **exp**)], [CREATE (eg-x in **range**)])), ...) ;
		→ (.., (DESTROY ((EVERY x **range**) **exp**)), ...) ;
[DES-E →]	T	(.., ([DESTROY (**subst** ug-x x **exp**)] AND [DESTROY (ug-x in **range**)]), ...) → ;
		(.., (DESTROY ((EXISTS x **range**) **exp**)), ...) → ;
[→ DES-E]	T	→ (.., ([DESTROY (**subst** ui-c x **exp**)] AND (DESTROY (ui-c in **range**)), ...) ;
		→ (.., (DESTROY ((EXISTS x **range**) **exp**)), ...) ;

if it is true in the world state, then the **destroy-action-of** the **patm-exp** replaces the DESTROY expression. If **patm-exp** is an object, then it is replaced by the actions needed to destroy the object. It may be noted that (DESTROY (x OR y)) is not the same as ((DESTROY x) OR (DESTROY y)); instead, it is (ASSERT ((DESTROY x) AND (DESTROY y))). This indicates that the actions needed to (DESTROY x) and (DESTROY y) should be created in parallel.

The destruction of a conjunction is the same as the conjunction of destructions. This also is the dual of the situation that occurred for the CREATE expressions, where the creation of a disjunction was the same as the disjunction of creations. For DESTROY expressions, using rule [DES-CNF], its argument is put in *conjunctive normal form*, (a conjunction of disjunctions) which is again the dual of the disjunctive normal form.* The [DES-OR] rule is always applied simultaneously to all the disjuncts in the disjunction.

The rule for [DES-U →] may be interpreted as follows:

(4.1). (DESTROY ((EVERY x **range**) **exp**)) IFF
(4.2). (DESTROY ((EVERY x)([x in **range**] IMPLIES **exp**)) IFF
(4.3). ((EXISTS x)(DESTROY ([x in **range**] IMPLIES **exp**))).

*As for example, ((x AND y) OR z) = ((x OR z) AND (y OR z)).

One gets [DES-U →] by applying ([E →], [DES-IMP]) in sequence to the expression (4.3) above, and [→ DES-U] by applying ([→ E], [DES-IMP]) to (4.3).* Similarly,

(4.4). (DESTROY ((EXISTS x **range**) **exp**)) IFF
(4.5). ((EVERY x)(DESTROY ([x in **range**] AND **exp**))),

and the [DES-E →] and [→ DES-E] rules are derived from the expression (4.5) by applying rules ([U →], [DES-AND]), and ([→ U], [DES-AND]), respectively.

The reduction rules for PREVENT are presented next in Table B8. Preventing an expression from becoming true is the same as first destroying its truth and then preventing its recreation. For simple expressions (i.e., expressions without any time qualifiers), **sexp**, this is expressed by the rule [PRR] in Table B8. If the argument of a PREVENT expression is a term, then the intention is to prevent the truth of the value of the term. This is expressed by the rule [PRR-TRM] in Table B8. The rule [PRR-TM] specifies that **tm-terms** may be moved inside the scope of PREVENT from outside. The reduction rules for timed expressions are stated next in Table B8 in rules [PRR-AB] (for after, before expressions) and [PRR-INT] (for between and during expressions). The pattern, **aft-bef** in [PRR-AB] will match 'after' or 'before,' and **dur-bet** will match 'during' or 'between.' In all cases, the PREVENT expression gets transformed to a conjunction of a DESTROY expression and the PREVENTion of its recreation. Again, note that (PREVENT (x OR y)) will not be the same as ((PREVENT x) OR (PREVENT y)).

Table B8—Reduction Rules for PREVENT

Name	Condition	Inference
[PRR]	T	([DESTROY **sexp**] AND [PREVENT (CREATE **sexp**)])
		(PREVENT **sexp**)
[PRR-TRM]	T	(PREVENT (**value-of term**))
		(PREVENT **term**)
[PRR-TM]	T	(PREVENT (**tm-term exp**))
		(**tm-term** (PREVENT **exp**))
[PRR-AB]	T	([DESTROY ((before **tm-term**) **sexp**)] AND [PREVENT (CREATE ((**aft-bef tm-term**) **sexp**)])
		(PREVENT ((**aft-bef tm-term**) **atm-exp**))
[PRR-INT]	T	([DESTROY ((before **tm-term-1**) **sexp**)] AND [PREVENT (CREATE ((before **tm-term-2**) **sexp**))])
		(PREVENT ((**dur-bet** (**tm-term-1 tm-term-2**)) **sexp**))

*The rules [E →], [→ E], [OR →], [→ OR], [→ AND], [AND →], [U →], and [→ U] were introduced in Section 3, and [→ ASRT] is defined in Table B9b.

The important point to note is that PREVENT has a semantics that is quite different from those of CREATE and DESTROY. Whereas CREATE and DESTROY are one-time operations, PREVENT requires a persistant action; there is need to prevent the recreation of the destroyed object. Also, note that (PREVENT (NOT exp)) will get transformed to:

([DESTROY (NOT exp)] AND [PREVENT (CREATE (NOT exp))])

by the [PRR] rule, and this in turn will get transformed to:

([CREATE exp] AND [PREVENT (DESTROY exp)])

via rules ([CRR-NOT], [DES-NOT]).

Similarly, there are inference rules for other operators, as well: rules for KEEP, SUPPORT, etc. I will not present here the rules for these, since we will not be using them in our example. Let me conclude this section with the inference rules for ASSERT. The ASSERT operator is used to make an expression true in a world state without having to go through its associated CREATE processes. The components of the statement are simply asserted into the world state. As mentioned in Section 3.3, this assertion will occur unconditionally if the ASSERT statement occurs on the left side of a sequent and conditionally if it is on the right.

The first five rules in Table B9a are quite straightforward. [ASRT-TM] in effect says that the time at which the assertion was made may be ignored in the reasoning process. The time of the assertion, however, is saved in the binding condition in a newly generated local variable, ei-t, for future use, if necessary. Since we are not interested here in analyzing who said what and when, I have chosen this simplification. In [ASRT-EL], the **not-true-part-of** function will return the part of **exp** that is not true in the world state. This is the dual of the **true-part-of** function mentioned earlier. The rest of the rules in the table need some explanation.

Table B9a—Reduction Rules for ASSERT

Name	Condition	Inference and Bindings
[ASRT-TM]	T	(ASSERT **exp**) binding: (ei-t = **tm-term**).
		(**tm-term** (ASSERT **exp**))
[ASRT-TRM]	T	(ASSERT (**value-of term**))
		(ASSERT **term**)
[ASRT-ASRT]	T	(ASSERT (..., **exp**, --, **exp**, ..))
		(ASSERT (..., (ASSERT (**exp**, --, **exp**)), ..))
[ASRT-OP]	T	(**operator-cd** (**exp** AND ... AND **exp**))
		(**operator-cd** (ASSERT (**exp**, ..., **exp**)))
[ASRT-EL]	(ISTRUE **exp**)	[(**true-part-of exp**) OR (ASSERT (**not-true-part-of exp**))]
		(ASSERT **exp**)

Table B9b—Reduction Rules for ASSERT (continued)

In the following, the condition 'not-true' = '(NOT (ISTRUE exp)),' where exp is the conjunction of the arguments of ASSERT		
[ASRT-EL →]	not-true	(**L-assert (atm-exp, ..., atm-exp)**) → ;
		(ASSERT (**atm-exp, ..., atm-exp**)) → ;
[→ ASRT-EL]	not-true	(**R-assert (atm-exp, ..., atm-exp)**) → ;
		→ (ASSERT (**atm-exp, ..., atm-exp**)) ;
[ASRT-DNF]	not-true	(ASSERT (**dnf conjunction**))
		(ASSERT **conjunction**)
[ASRT-U →]	not-true	(ASSERT (.., [(ug-x in **range**) IMPLIES (**subst** ug-x x **exp**)], ...)) → ;
		(ASSERT (.., ((EVERY x **range**) **exp**), ...)) → ;
[→ ASRT-U]	not-true	→ (ASSERT (.., [(ui-c in **range**) IMPLIES (**subst** ui-c x **exp**)], ...)) ;
		→ (ASSERT (.., ((EVERY x **range**) **exp**), ...)) ;
[ASRT-E →]	not-true	(ASSERT (.., (**subst** ei-c x exp)), (ei-c in **range**), ...)) →;
		(ASSERT (.., ((EXISTS x **range**) **exp**), ...)) → ;
[→ ASRT-E]	not-true	→ (ASSERT (.., (**subst** eg-x x exp)), (eg-x in **range**), ...)) ;
		→ (ASSERT (.., ((EXISTS x **range**) **exp**), ...)) ;
[ASRT-OR]	*not-true*	([ASSERT **exp-1**] OR [ASSERT **exp-2**])
		[ASSERT (**exp-1** OR **exp-2**)]
[ASRT-AND]	not-true	(ASSERT (.., **exp**, ...))
		(ASSERT(..AND **exp** AND...))
[ASRT-IMP]	not-true	(ASSERT (.., ([NOT **exp-1**] OR **exp-2**), ...))
		(ASSERT (.., (**exp-1** IMPLIES **exp-2**), ...))

In its simplest form, the argument of an ASSERT expression is a series of expressions separated by commas. As indicated by the [ASRT-AND], these commas are interpreted within the ASSERT as AND. Inside an ASSERT, the AND's may be replaced by commas. This is called a parallel ASSERT. If all the arguments of an ASSERT are atomic expressions, then for each atomic expression, its truth value is first set in the world state to T if it is positive, and to F if it is negated. Only after setting the truth values for all the atomic expressions in the ASSERT statement will CK-LOG check the world state for the consistency. Thus, in effect, they are all asserted in parallel. This is what happens when the **L-assert** function is executed in [ASRT-EL →] rule. The **L-assert** function will return the conjunction of all the atomic expressions that were successfully asserted into the world state. It will return NIL if the assertion is not successful. The **L-assert** happens unconditionally, when the ASSERT statement is on the left side of a sequent.

If it is on the right side of a sequent, then the **R-assert** function is used. This function will first check whether the arguments of the ASSERT expression satisfy the matching requirements described in

Section 3.3. If all the atoms in the ASSERT expression are matched successfully, then the assertion will be performed using the **L-assert** function. If matches could not be found for some or all of the arguments of the assertion, then the system will present to the user the list of atomic expressions for which matches could not be found in the sequent. At this point, the user may either advise the system to reject the assertion, or ask the system to accept the atoms even though no matches exist for them in the sequent. The conjunction of these atoms will then be introduced by the system as a *hypothesis problem* associated with the sequent in which the ASSERT expression appeared.

[ASRT-DNF] puts the argument of an ASSERT into disjunctive normal form, if the argument is a **conjunction** (which can be a series of expressions separated by commas). Notice that in the rules above the pattern '(..,exp, ...)' indicates one or more expressions, exp, separated by commas (or AND, or OR, as the case may be). [ASRT-OR] converts the ASSERT of a disjunction to a disjunction of ASSERTS. If the argument is quantified expression, then ASRT-U and ASRT-E are used to eliminate the quantifiers. They introduce the new variables, ug-x, ui-c, eg-x, and ei-c, depending on the various cases. The binding conditions specify their ranges and indicate that these were created inside the scope of an ASSERT statement. These rules allow the quantifier elimination rules to be applied even when the quantifiers are inside the scope of an ASSERT. In [ASRT-OP] rule, the pattern, **operator-cd** is the same as **operator** with CREATE and DESTROY excluded. I have assumed throughout that negations will appear only with atomic expressions.

Among the rules presented above and those presented in Section 3, the various rules for NOT, AND, OR, IMP, and IFF expressions, and the rules for reducing timed expressions, are called *propositional rules*. The rules for quantified expressions are called *quantifier elimination rules*. In applying a rule to a given sequent, there are two choices to be made: the expression in the sequent to which a rule is to be applied (note that in general a sequent will have several expressions), and the rule to be applied to the chosen expression. The choice of an expression in a sequent might in certain cases be determined by the ordering of time expressions associated with them (an example of this occurs in seq-6 in Section 5.1). If there are several choices available for expressions, then either one may be chosen arbitrarily, or the user may specify to the system the choice to be made.

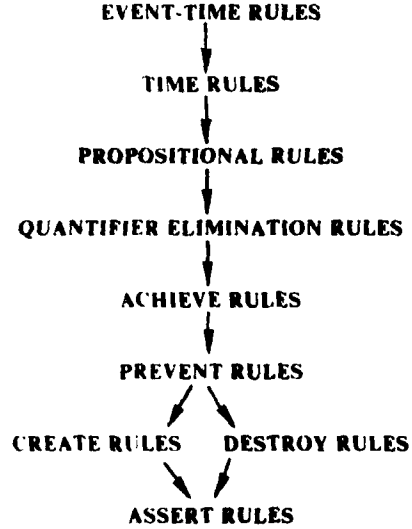


Fig. B1—The order of application of inference rules

Once the expression is chosen, the rule (set of rules) to be applied to it will usually be unique. These rules are applied to the chosen expression in the following order: first, all the applicable event-time rules (Table B4) are applied, then all the applicable time rules (Table B5) are applied, then the applicable propositional rules (Section 3) are applied, then the quantifier elimination rules (Section 3), then the ACHIEVE rule (Table B2), then the PREVENT rules (care should be taken here to prevent an endless loop because of recursion in the PREVENT rules), then the CREATE or DESTROY rules, and finally the ASSERT rules. This is diagramed in Fig. B1. Section 5 illustrates the way these rules are used in a problem-solving process.

GPO 918-478

END

FILMED

12-85

DTIC